Universidade de Aveiro
Departamento de Eletrónica,
Telecomunicações e Informática
2017

Bernardo Menezes Rodrigues Ferreira

Automação de Edifícios com Tolerância a Falhas

Building Automation with Failure Tolerance

Universidade de Aveiro
2017

Departamento de Eletrónica,
Telecomunicações e Informática

Bernardo Menezes Rodrigues Ferreira

Automação de Edifícios com Tolerância a Falhas

Building Automation with Failure Tolerance

Dissertação apresentada à Universidade de Aveiro para cumprimento dos requisitos necessários à obtenção do grau de Mestre em Engenharia de Computadores e Telemática, realizada sob a orientação científica do Doutor João Paulo Silva Barraca, Professor auxiliar do Departamento de Eletrónica, Telecomunicações e Informática da Universidade de Aveiro, e do Mestre Nuno Lourenço da empresa Think Control.

o júri / the jury

presidente / president

Prof. Doutor Rui Luís Andrade Aguiar
Professor Catedrático da Universidade de Aveiro

vogais / examiners committee

Prof. Doutor Jorge Sá Silva
Professor Auxiliar com Agregação da Universidade de Coimbra

Prof. Doutor João Paulo Barraca
Professor Auxiliar da Universidade de Aveiro

agradecimentos /
acknowledgements

Gostaria de deixar aqui um agradecimento ao meu orientador, Professor Doutor João Paulo Barraca, e coorientador, Nuno Lourenço, pelo apoio e acompanhamento dado ao longo da elaboração desta dissertação.
Quero deixar ainda, um agradecimento especial à minha família e aos meus amigos pelo incançável apoio e motivação ao longo deste tempo.
A todos um enorme obrigado.

Palavras Chave

Internet das Coisas, Redes de Sensores, Automação, Processamento de Eventos Complexos.

Resumo

Esta dissertação enquadra-se no projecto SmartLighting e tem como objectivo criar uma solução energeticamente eficiente para edifícios e espaços inteligentes. Numa primeira fase, esta dissertação apresenta uma revisão das soluções existentes de automação de edifícios e, posteriormente, propõe uma solução baseada em princípios da Internet das Coisas e sistemas de processamento complexo de eventos, capaz de criar um ambiente inteligente, autónomo e resiliente a falhas. O foco do trabalho está na criação de um software leve para ser colocado em dispositivos com pouca capacidade de processamento de modo a poderem, não só ser um meio para comunicação com dispositivos inteligentes, mas também habilitados para oferecer capacidades de processamento de eventos em casos de emergência.

Keywords

Internet of Things, Wireless Sensor Networks, Automation, Complex Event Processing.

Abstract

This dissertation was was done within the scope of the SmartLighting project and aims to create an energy efficient solution for buildings and smart spaces. In a first phase, this dissertation presents a review of existing building automation solutions and later proposes a solution based on Internet of Things (IoT) principles and Complex Event Processing (CEP) systems, capable of creating a smart, autonomous and fail resilient environment. The focus of the work is on creating a lightweight software to be placed on devices with low processing capacity so that they can not only be a means of communicating with intelligent devices but also enabled to provide event processing capabilities in cases of emergency.

Contents

List of Figures

List of Tables

Acronyms

6LoWPAN	IPv6 over Low-Power Wireless Personal Area Networks
AMQP	Advanced Message Queue Protocol
BAS	Building Automation System
BACnet	Building Automation and Control Networks
BLE	Bluetooth Low Energy
BM	Building Manager
BMS	Building Management System
SIG	Bluetooth Special Interest Group
CEP	Complex Event Processing
CFL	Compact Fluorescent
CoAP	Constrained Application Protocol
DALI	Digital Addressable Lighting Interface
GM	Gateway Manager platform
GAE	Gateway Automation Engine
IEEE	Institute of Electrical and Electronics Engineers
IoT	Internet of Things
IPv6	Internet Protocol version 6
IT	Instituto de Telecomunicações
KNX	Konnex
LED	Light-Emitting Diode
LonWorks	Local Operating Network
HVAC	Heating Ventilation and Air Conditioning
M2M	Machine-to-Machine
mDNS	Multicast Domain Name System
MQTT	Message Queue Telemetry Transport
OSI	Open System Interconnection
QoS	Quality of Service
REST	Representational State Transfer
SIG	Bluetooth Special Interest Group
SoC	System on a Chip
TCP	Transmission Control Protocol
TMR	Triple Modular Redundancy
UDP	User Datagram Protocol
XML	eXtensible Markup Language
XMPP	Extensible Messaging and Presence Protocol
WSN	Wireless Sensor Network

CHAPTER 1

Introduction

Over the last decades, there has been a rising dependence on information and communication technology, cross-cutting every sector. The fast growth of society is directly related to technological development as it aims to solve everyday problems. Each person, company or industry rely closely on technology.

However, this dependence comes at the cost of energy consumption, which is an essential and expensive requirement. This fact, urged society into finding affordable and environmentally friendly ways of generating energy, such as renewable energies. Yet, even though this type of energy producers have undesired effects on landscape preservation the solution for energy problems lies in reducing as much as possible energy consumption around the globe, and not just trying to find cheap ways of producing it.

The amount of energy consumed in the non-residential buildings is a huge part of the worldwide energy consume [1]. This fact has been acting as an influence for finding efficient ways of reducing the energy consumption in this sector. There are already several solutions in BAS's (Building Automation Systems), which achieve good results in automation mechanisms to reduce energy consumption in buildings. However, the majority of BAS solutions are based on proprietary implementations that require most, if not all, of the hardware, installation and management to be done by the same vendor. Even when applying industry standards, device interoperability is often limited. Moreover, as a strong downside these systems typically do not allow complex correlation between events. Simple action-reaction systems are not enough.

The rise of the Internet of Things (IoT) paradigm has enabled devices to share large amounts of data, that can be processed to extract additional information. This can be used to improve the automation of real world tasks. This concept aims to improve comfort and decrease costs, which are the main goals of smart buildings.

The work presented in this document fits this approach, as it aims to create a smart environment where the information of connected devices is gathered and processed in order to, not only achieve real time acting over the occurring events in a building, but also making sure that the whole systems can work and adapt when some of its components fail.

1.1 MOTIVATION

This dissertation is part of a research project of Instituto de Telecomunicações (IT) of Aveiro named "SmartLighting". The main goal of this project is to endorse the second building of this institute (IT2) with a network of sensors and actuators in order to obtain an energy efficient solution, reducing operational costs and increasing comfort of occupants. Despite being a relatively recent building, it lacks the support for intelligence and automation systems, and this fact entailed the appearance of the SmartLighting project.

Both this dissertation and project have the support of Think Control, an engineering consultancy company in the area of efficient lighting systems and building automation.

1.2 OBJECTIVES

The aim of this dissertation is not only, improve the energy efficiency and reduce operational costs of a building while applying IoT concepts, but also, create a secure and reliable environment resistant to fails in every element of its whole architecture.

It was expected that this dissertation built upon prior work done for this project that included a network of gateways connected to sensors and actuators [2]. On top of that the goal was to achieve a decentralized solution, removing single points of failure and also enabling faster reaction times to core events and unexpected system failures.

As stated before, all new features needed to integrate fully with the actual state of SmartLighting project to grant continuity of the research work and allow additional developments in future dissertations.

1.3 CONTRIBUTIONS

This dissertation contributes with a decentralized solution of gateways that are able to react quickly to unexpected events, such as the CEP system failure, but also to ensure that basic building automation operations are processed as quickly as possible.

One of the most important aspect that has been taken into account was the use of lightweight technologies to ensure the whole solution could fit in a resource limited microcontroller device acting as gateway.

Finally, it a demonstrator was implemented consisting in a dashboard to control and observe in real time the state of all registered gateways in the system. This demonstrator and prior work done in SmartLighting project was presented at the "Students and Teachers@DETI 2017" event, where dissertations, personal projects and course projects were presented publicly to the community.

1.4 STRUCTURE

Following this introduction chapter, the remainder of the document is organized as follows.

- **Chapter 2:** presentation of the state of the art, including key concepts for the presented theme. These will focus on building automation, failure handling, Internet of Things and Complex Event Processing.
- **Chapter 3:** introduction to SmartLighting project and presentation of use cases and requirements for this solution. Also, an explanation of architecture and components is provided.
- **Chapter 4:** description of the chosen implementation for the architecture presented in Chapter 3 and explanation of the implementation decisions taken.
- **Chapter 5:** presentation and analysis of the results obtained from testing, including test methodology, and analysis of results under a feasibility perspective.
- **Chapter 6:** concluding remarks regarding the achieved goals for the developed solution and presentation of potential future steps.

CHAPTER 2

State of the Art

This chapter aims to provide a description and overview of important concepts of Building Automation Systems, Internet of Things, fault tolerance in component-based systems, as well as event processing. For each topic there is a review of the different technologies and protocols, how they interact and how they differ from each other.

All of the topics addressed in this chapter contributed to the chosen technologies for this solution implementation.

2.1 Building Automation System

A Building Automation System consists in the automatic control and monitoring of building services such as lighting, heating, ventilation, air conditioning, amongst others. The main purpose of BAS is to improve the comfort of the building's occupants, while significantly decreasing the operational costs and energy consumption. This is achieved using the information gathered by a wide set of sensors, that is used to control building's equipment behaviour [3]. This information can also be used to prevent and react quickly to emergency situations, allowing to resolve them with minimum or even non human intervention.

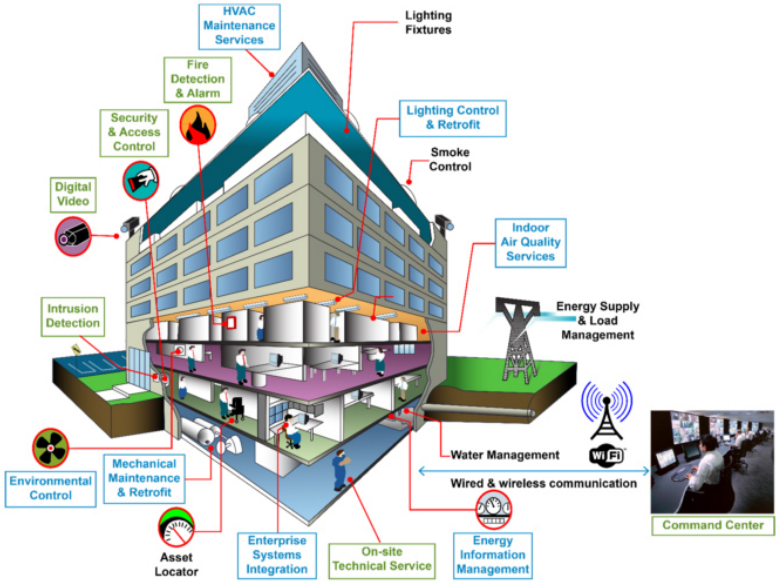

Figure 2.1: Building Automation System schematic. Adapted from [4].

The biggest problem in building automation is market segmentation, where manufacturers from different segments created, over the years, either proprietary solutions or application specific protocols and communication standards. This lead to a scenario where integrating all building automation features into a single concentrated system is a quite difficult and complex task. The most recurrent scenario includes several application specific automation systems which are monitored by a higher level Building Management System (BMS).

In the past it was quite common for each vendor to have proprietary specifications. However, as BAS and BMS systems became widespread, market competition and request for interoperability lead manufacturers to disclosing their specifications in an attempt to secure a larger market share. This led to the rise of new standards, often as the result of a combination of several vendor proprietary specifications. Currently, the most common communication and management standard used in BMS is the Building Automation and Control Networks (BACnet) [5], Konnex (KNX) [6], Local Operating Network (LonWorks) [7], and Digital Addressable Lighting Interface (DALI) [8]. Each one of them as an important role at different levels of a building management and automation system. As shown in Figure 2.2, those levels are Management System Level, Automation Level and Field Level [9]. The all set of devices, sensors or actuators, compose the field layer. The Automation level is responsible for all the logic and hardware needed to control those devices and the Management Level consists in the control and management of the system and the reactions to events present in the gathered data.

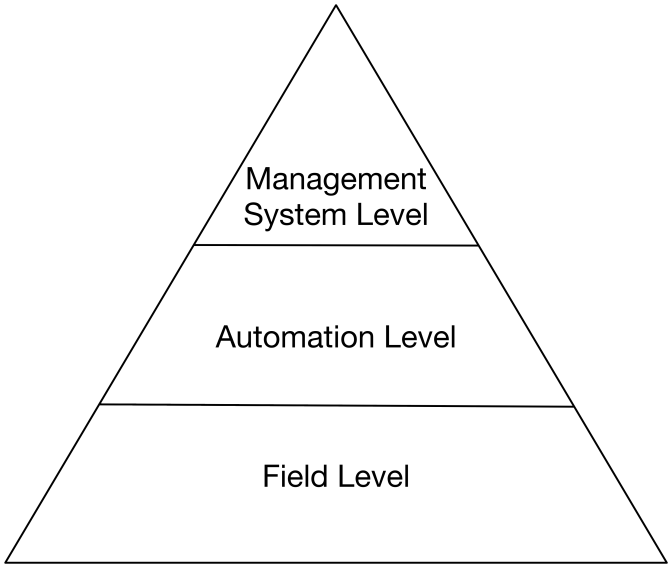

Figure 2.2: Building Automation System hierarchy.

BACnet fits within the Managemet System level, allowing monitoring different automation systems, while KNX, LonWorks and DALI are Automation level protocols [10]. Regarding DALI, it has its own Field level specification, which is a wired solution specific for lighting. KNX and LonWorks allow different Field level protocols, having the first one a wireless solution, KNX RF [11].

2.1.1 BAS solutions

Over the years a large number of companies stepped forward to offer solutions for reducing energy consumption in buildings. Companies like Siemens, which provides three different BAS/BMS systems, Honeywell, and Johnson Controls, which claims to be market leader, offer services, using, most of the times, the formerly addressed open standard protocols.

However, these products have some common drawbacks: most of them are highly costly, which ends up to be a problem for medium to small companies to afford, and also, are not open source and don't allow for building managers to install and integrate new devices and feature at will.

With the rising of IoT, there has been an increasingly amount of novelty solutions that use high scalable technologies, well known communication standards, such as IP, and use lightweight devices with low power consumption. This concepts can also be applied to BAS/BMS and will be addressed over the next section.

2.2 Internet of Things (IoT)

Internet of Things is a concept that has emerged in recent years. It entails that in addition to the devices we usually connect to the Internet, like smartphones or personal computers, a new diversity of devices will also be connected, forming a vast network of smart objects. Devices like vehicles or normal household appliances will be equipped with sensors and actuators that will be able to gather data, giving people and processing systems the information to take better and more valuable decisions [12]. Also, with controllers and the sensing information, these devices can perform certain tasks without human intervention. There was already a concept, used in telemetry, industrial and automation systems, Machine-to-Machine (M2M), which enabled the exchange of information between machines, in order to perform tasks and actions without human intervention. Although both of the concepts goals remain alike, i.e. enable communication between machines, M2M should be considered as a subset of IoT, as it lacks in the creation of meaning to the information exchanges. IoT is a wide web of not only M2M networks but also, a vast set of applications that enables, as stated in the previous section, automatic decision making and data analytics.

The concept of IoT was first introduced by Kevin Ashton [13] in 1999, where he stated his idea of having computers gathering information from things around us, that humans couldn't do by themselves, to give a new perspective to people on how to reduce waste, loss and cost. For this author, we needed to give computers the power to gather information with their own means, so they could sense the world for themselves and overcome the limited time, attention and accuracy humans have.

In the recent years, with the technological advances made in electronics, devices became smaller, low powered and more affordable, giving a boost to the integration of this concept in our society. According to [14], the turning point for IoT would be when there were more devices connected to the Internet than people. This point was reached somewhere between 2008 and 2009, and as can be seen in Figure 2.3, this numbers will grow exponentially in the next years and can even reach over a 6 to 1 factor between connected devices and the world population.

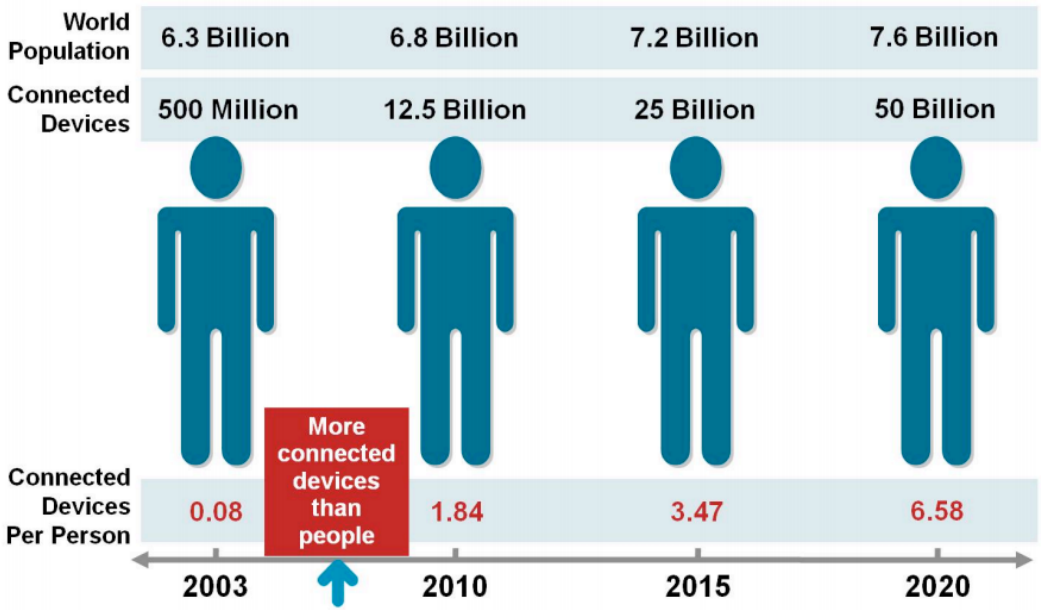

Figure 2.3: Estimated growth of connected devices [14].

Nevertheless, the growth of IoT still faces some challenges. As the authors of [15] state, with the growing number of devices being connected to the Internet, Internet Protocol version 6 (IPv6) is going to be crucial to cope with billions of new devices that will require a unique IP address. Nevertheless, IoT remains with no agreements on what standards should be used, which is a problem since it hampers the communication between devices

The IoT concept, according to [14] will be a very important evolution to the Internet, giving the ability to create new applications to improve people's lives. The sharing of knowledge and wisdom has always been a key factor for human evolution, and now, with IoT and the help of smart devices that sense, collect and share information, we can create new ways to enhance people's comfort and well-being.

IoT will have an important role in a vast number of sectors, and building automation is an example of that. The IoT's rapid growth is giving building managers a wide variety of new technology choices. Darcy Otis, director of analytics and fault detection for the building technologies division at Siemens states that IoT will give BAS more information and better tools to perform its core functions [16].

2.2.1 Wireless Sensor Network (WSN)

WSN is a network of distributed devices, sensors and/or actuators, interconnected by a wireless medium. As stated in [17], this network of nodes sense and control the

environment cooperatively, enabling the interaction between persons or computers and the environment around them.

Nowadays WSNs usually include sensors, actuators and gateways. Sensors are devices capable of detecting changes in environmental variables and supply that information to other nodes. These devices can measure variables like luminous intensity, temperature, humidity, pressure, motion and a wide variety of other environmental data. Actuators, in the other hand, are devices that control mechanisms to alter the physical environment state and perform these actions based on input data. For instance, an actuator can be programmed to turn on or off an air conditioner based on a input trigger set off by the temperature in a room. Lastly, gateways are devices connected to sensors and actuators, responsible for orchestrate and interconnect this devices. Gateways can also be programmed to, based on input data gathered by sensors and a set of pre-loaded rules, trigger an actuator to act on some physical environment element, autonomously.

The connection between these three elements form a WSN, which is the primary enabler of solutions such as building automation. Also, WSNs allied with IoT protocols and standards, can truly create smart environments. Over the following sections these IoT protocols will be discussed.

2.2.2 WIRELESS COMMUNICATION PROTOCOLS

In this section some of the most used communication protocols in IoT are presented. Since IoT devices often require wireless networking capabilities and to be battery-powered, the main focus for this section will be wireless protocols with low power usage.

When talking about wireless communication, the protocol that first comes to mind is IEEE 802.11, better known as Wi-Fi. This wireless standard was introduced in 1997 by the Institute of Electrical and Electronics Engineers (IEEE) [18], and nowadays is used across laptops, phones and other media devices. Although it has an important role for this kind of technological equipment, IEEE 802.11 has never been widely used for communication between IoT devices. The reason for that are WiFi's power requirements, often making it unsustainable for smart devices to use.

Another important and well known wireless protocol for communications between devices is Bluetooth. This standard was created in 1999 by Bluetooth Special Interest Group (SIG) and in 2016 reached its fifth version [19]. Bluetooth is widely used for connecting smartphones to other bluetooth enabled devices, such as wearables, cars

and others, making it one of the most used close range communication protocol.

Both Bluetooth and Wi-Fi standards have polished and mature specifications, making them reliable communication protocols. However these standards are not suited for low capabilities devices since they were not created with restrictions like low power consumption and limited resources in mind. Over the next sections, will be presented a variety of protocols, more suited for this kind of devices.

2.2.2.1 IEEE 802.15.4

The IEEE 802.15.4 standard was introduced by IEEE and its intent was to target Low Rate WPANs (LR-WPANs) [20]. This protocol specifies the two lowest layers of the OSI model, represented in Figure 2.4, and it is branded for its simplicity, low data rates and battery saving [21].

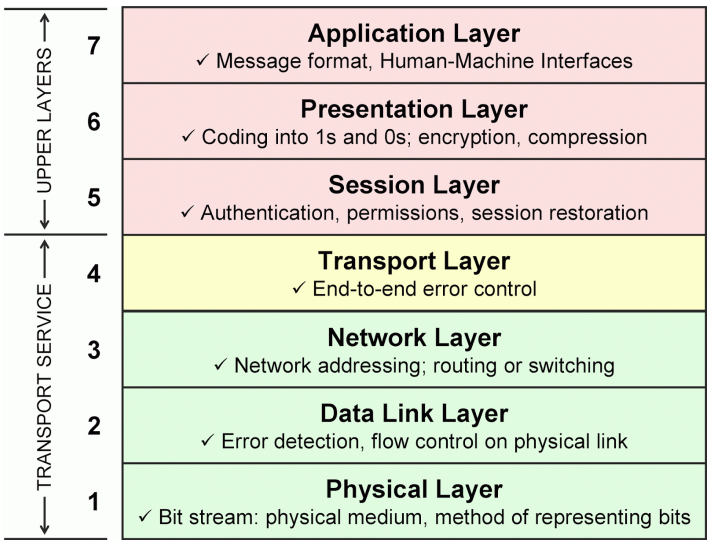

Figure 2.4: Open System Interconnection (OSI) model [22].

These characteristics are what make this standard quite significant for IoT. Since IoT sensors and actuators have many battery constrains, using a protocol that has these concepts in mind, make IEEE 802.15.4 a suitable standard for communications within WSNs.

IEEE 802.15.4 is also the baseline for other IoT communication protocols such as ZigBee, which will be covered in a section below.

2.2.2.2 6LoWPAN

IPv6 over Low-Power Wireless Personal Area Networks (6LoWPAN) is a standard presented by Internet Engineering Task Force (IETF), and its main objective was to implement IPv6 over low power networks, granting the possibility for IoT devices to be reached through the Internet using a unique address. Also this would allow the use of some IPv6 important features such as Quality of Service (QoS), mobility and multicasting.

6LoWPAN uses the previously addressed standard, IEEE 802.15.4, for the two lowest layers of the OSI model, shown in Figure 2.4. Since the network layer protocol was needed to comply with those two lower layers, and the requirements for an IPv6 based protocol didn't match the constrained IEEE 802.15.4 standard, 6LoWPAN implemented mechanisms to efficiently transport IPv6 packets within small link layer frames [23].

In order to use 6LoWPAN in a IPv6 network, it is only needed a low complexity edge router to route the traffic from IPv6 to 6LoWPAN, as shown in Figure 2.5.

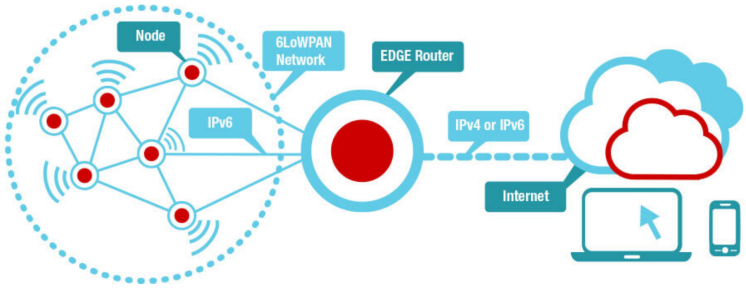

Figure 2.5: Connection between a 6LoWPAN and an IPv6 network [24].

To sum up, with the proliferating market of IoT devices, the use of IPv6 is unavoidable, and the fact that 6LoWPAN offers a low power and low complexity way of addressing this problem, makes this protocol a viable solution for its use in IoT environments [25].

2.2.2.3 ZigBee

ZigBee is a standard introduced by ZigBee Alliance, released to the public in June 2005. ZigBee is built on top of the IEEE 802.15.4 standard, addressed early, and it specifies the its own implementation of upper layers [26]. As shown in Figure 2.6, ZigBee provides on the Application, a ZigBee Cluster Library (ZCL) that maps commands to be used by developers when defining application profiles.

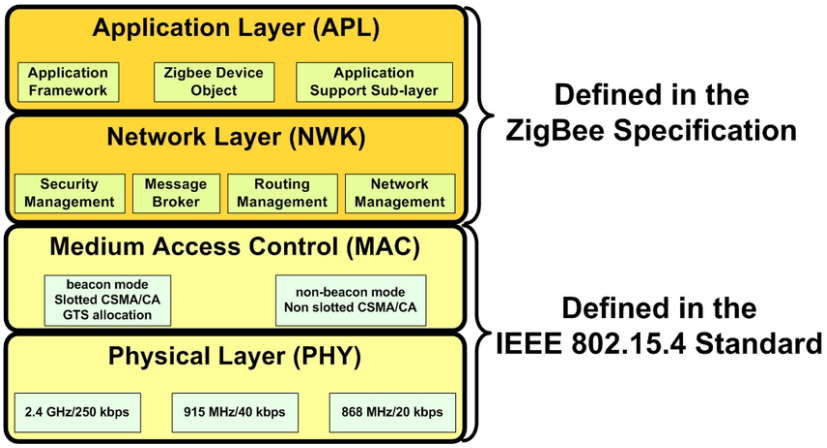

Figure 2.6: ZigBee protocol stack [27].

Although other standards have been emerging on the IoT constrained devices market, ZigBee widely dominates this market as it has been targeting it since the beginning, as a very low power and low latency standard. Although ZigBee is not suitable for long ranges and has low data transfer speeds, a single master node is able to control 254 devices such as light bulbs and light switches [25].

Early this year, ZigBee Alliance unveiled dotdot[1], an application layer protocol that aims to reduce the fragmentation in IoT by offering a common language between all IoT devices on any network. Due to a cooperation between ZigBee Allience and Thread Group, a company that created Thread[2], which is an IPv6 based protocol, that like ZigBee is based on IEEE 802.15.4, dotdot is set to launch its first certified products later this year.

2.2.2.4 BLUETOOTH LOW ENERGY

Bluetooth SIG announced, in 2009, Bluetooth 4.0 including Bluetooth Low Energy (BLE) as a way to address the low power constrained devices that emerged with the rising growth of IoT.

BLE was a radical departure from the Classic Bluetooth and there was no compatibility between the two. BLE wouldn't be compatible with most of the widely adopted Bluetooth protocol that most smartphones and tablets used. This fact forced the emerging of two new classes of bluetooth devices: Bluetooth Smart and Bluetooth Smart Ready, as shown in Figure 2.7 [28]. The first one, known as single-mode device, is only capable of communicate using BLE, being used in sensors (presence sensor, temperature

[1]https://speakdotdot.com/
[2]https://threadgroup.org/

sensors, etc.) or other types of battery-operated and low processing accessories. The second, also known as dual-mode Bluetooth, is capable of communicating with both BLE devices and classic Bluetooth devices, being more suitable for smartphones, IoT gateways and other more powerful devices.

Figure 2.7: Product examples of Bluetooth single and dual-mode modules. Adapted from [28].

BLE is suitable to low-energy devices because it adopts a client/server model where devices are in a sleep state and periodically send advertisement messages for clients, such as a smartphone or a gateway, discover it and begin a connection as shown in Figure 2.8. When connected, the clients can ask for data in a pre configured interval [28].

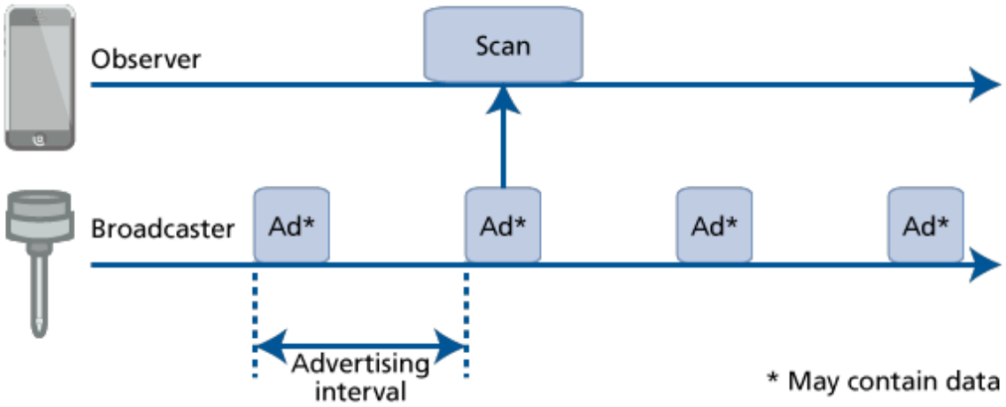

Figure 2.8: BLE discover of devices. Adapted from [28].

More recently, Bluetooth SIG introduced support for IPv6 networking in Bluetooth 4.2 and Bluetooth 5.0 taking advantage of the 6LoWPAN standard.

To sum up, BLE is one of the best solutions for communications between IoT devices due to its low energy consumption and acceptable data transfer rates.

2.2.2.5 WIRELESS COMMUNICATION PROTOCOLS COMPARISON

In this section is presented a comparison between the addressed wireless communication protocols, adding two more wireless technologies: Wi-Fi HaLow [29] and Thread.

Briefly, IEEE 802.11ah, also known as Wi-Fi HaLow, is a recently published standard that aims to bring Wi-Fi closer to IoT solutions. This is a strong possibility since Wi-Fi HaLow uses a lower band than the classic Wi-Fi, which confer to it nearly twice the range while also granting a more robust connection in any kind of environments [29]. Wi-Fi HaLow is also an energy-efficient protocol that, allied to the benefits of Wi-Fi, such as government-grade security and easy setup, can become an important player in the wireless communication protocols for IoT.

Regarding to Thread standard, as addressed before, is an IPv6 based protocol, that like ZigBee is based on IEEE 802.15.4. Although this protocol was created for smart houses, Thread Group is expanding its connectivity to the buildings sector.

These standards, although not fully or already available, have great potential for future adoption and therefore will also be considered.

	Frequencies	**Max Data Rate**	**Max Range**	**IP Support**
BLE	2.4 GHz	1 Mbps	100 m	Yes
ZigBee	2.4 GHz 915 MHz (US) 868 MHz (EU)	250 Kbps	70 m	Yes
Wi-Fi HaLow	865 MHz (EU) 920 MHz (US)	150 Kbps	1 Km	Yes
Thread	2.4 GHz	250 Kbps	200 m	Yes

Table 2.1: Wireless Communication Protocols Comparison

Regarding the protocols addressed in the previous sections, BLE and ZigBee, they are both suitable for a building automation solution. However, BLE have higher data rates than ZigBee, while achieving highly efficient power saving. Also, another advantage of BLE over other protocols is its widespread integration in smartphones, tablets and even laptops, with a large share of them already equipped with Bluetooth Smart Ready. This makes it better suitable for some IoT applications such as health care and building automation, where the user's smartphone can be an integral part of the system, being able to interact with the other devices.

2.2.3 APPLICATION PROTOCOLS

As previously stated, the fast growth of IoT devices motivated the search for lightweight protocols in order to enable constrained devices to efficiently communicate with other nodes.

In this section will be presented protocols that address the application layer of the OSI model, shown in Figure 2.4, and are suitable to be used in IoT, are presented. Since IoT solutions address a huge amount of different scenarios, with different requirements, bellow protocols that have different features will be presented, and the strengths and weak points of each will be evaluated.

2.2.3.1 MQTT

Message Queue Telemetry Transport (MQTT) is a M2M communication protocol introduced by IBM in 1999, but was only standardized in 2013 as OASIS [15]. MQTT follows a publish/subscribe client/server model and, due to its simplicity, it is suitable to be used in constrained devices.

The publish/subscribe model features three components: a broker, publishers and subscribers. In the scenario shown on Figure 2.9, the central piece, the server, is a MQTT Broker, and the other components such as data gatherers and data consumers are clients that connect to this broker. Clients are able to publish and/or subscribe to topics on the MQTT broker, and its task is to route messages from publishers, to the corresponding subscribers of each topic [15].

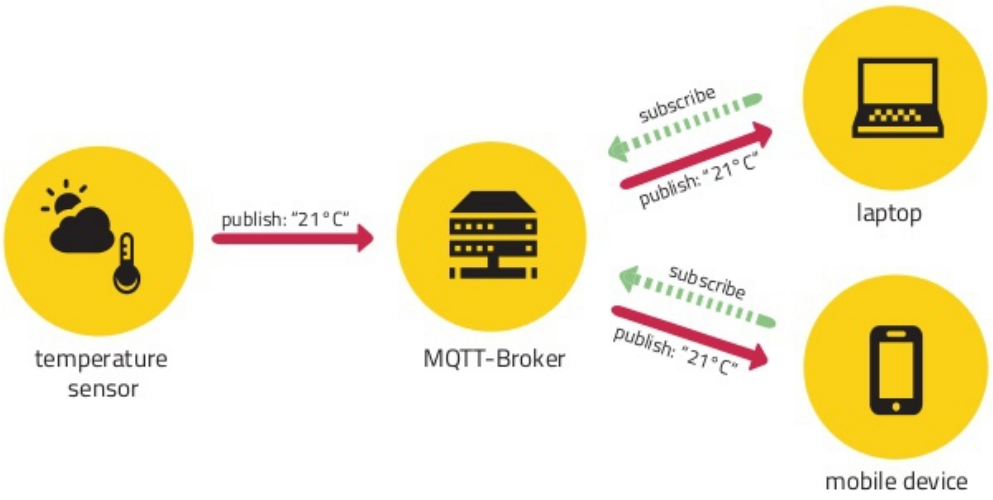

Figure 2.9: MQTT protocol architecture [30].

Taking as example the case illustrated in Figure 2.9, a temperature sensor publishes data on some topic and the MQTT broker forwards this messages to the clients subscribed to that temperature topic. MQTT offers the possibility of four different communication topologies: one-to-one, many-to-one, one-to-many, many-to-many [31]. The advantage of this protocol is that publishers do not need to send the same information to every client nor need to be aware of its presence. This makes the protocol well suited to be used in contexts where devices need to be as efficient as possible, which is the case of IoT environments.

Regarding the guarantees of a successful message delivery, MQTT has three levels of Quality of Service (QoS): at most once (0), at least once (1) or exactly once (2) [32]. The level 0 of QoS, also known as "Fire and Forget" is the most basic one, since the message is sent only one time to the subscribers. It has no fail mechanisms, so if, due to a network error, the message does not reach its target, is not resent by the broker. This level of QoS is suited when occasional unsuccessfully message delivers do not jeopardy system operation or when the broker-clients connection is stable enough. With level 1 of QoS, the sender stores the published message in cache and waits for the acknowledge message. If the acknowledge is not received within a pre-programmed time interval, the message is sent again. The drawback of this level os QoS is that a message can be received multiple times if the acknowledge does not reach the sender. Lastly, QoS level 2 grants that a message is delivered exactly one time, however this process introduces a greater overhead, as it takes longer than any other level of QoS.

Other important features of MQTT are the last will message, where a client can register a message to be delivered to a specific topic in the case of a lost of connection between the broker and the client, and the security concerns, in which an authentication method is provided, using a username and password. Also, regarding security, the payload of the message can be encrypted with SSL/TLS.

To sum, MQTT is an important Application layer protocol and very suitable to be used in constrained networks and devices as a powerful communications standard.

2.2.3.2 COAP

Constrained Application Protocol (CoAP) is an application protocol created by IETF with the objective of providing Representational State Transfer (REST) functionalities to constrained devices and networks. REST is a standard used in communications between computer systems on the Internet. Although this principle is a well-known and adopted standard for HTTP, when talking about IoT, whose devices are energy-wise constrained, this standard presents a high overhead [33]. CoAP comes to resolve this

drawback by presenting a lightweight RESTful solution, following a request/response pattern, with the constrained devices market in mind.

As shown in Figure 2.10, CoAP, unlike HTTP, uses User Datagram Protocol (UDP) in the underlying layer of its architecture. This fact makes CoAP communication more efficient as packets are sent without handshaking, or congestion control mechanisms that would introduce more overhead, as used by Transmission Control Protocol (TCP). The communication reliability is implemented by CoAP lightweight internal mechanisms [33].

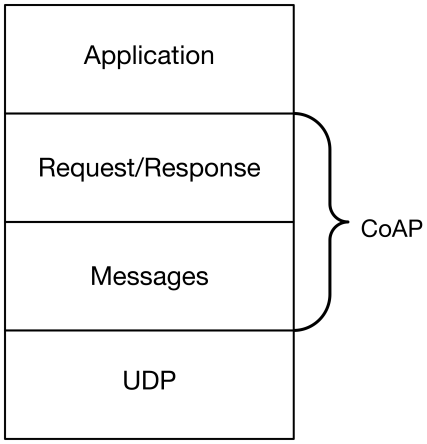

Figure 2.10: CoAP protocol architecture.

As is also shown in Figure 2.10, CoAP is divided into two sub-layers: request/response and messages. The messages layer is responsible for message processes such as fail and duplication handling, since UDP, as stated before, does not have reliability mechanisms [33]. The request/response layer is responsible for enabling the RESTful interaction by offering well known methods such as GET, POST, PUT and DELETE to interchange data [34].

CoAP has also an "observe" resource that enables clients, through a publish/subscribe mechanism, to observe a resource and when there is a change, the client is notified by the server [34]. All CoAP features make this protocol also very attractive for some use cases in an IoT environment.

2.2.3.3 AMQP

Advanced Message Queue Protocol (AMQP) is a communication protocol, initially created aiming at the financial sector, that, like MQTT, runs over TCP and adopts a publish/subscribe architecture. The use of this protocol for financial purposes is due to

the fact that AMQP has powerful mechanisms that grant high reliability and scalability that are crucial for that sector.

AMQP architecture is composed, as shown in Figure 2.11 by three elements: producers, clients and the broker, being the last one divided in two components, the exchange and topic queues. The producers publish messages in a specific topic and the exchange module is responsible for depositing those messages in the correct topic queue. Clients are connected to the queues of specific topics and then have access to the published messages in those queues [33]. This system allows that instead of the broker having to send the same message to multiple clients subscribed to a topic, the message is only sent to the respective queue. Also, it allows clients to be offline at the time the message is sent to the queue, since it is stored there until it is consumed [15].

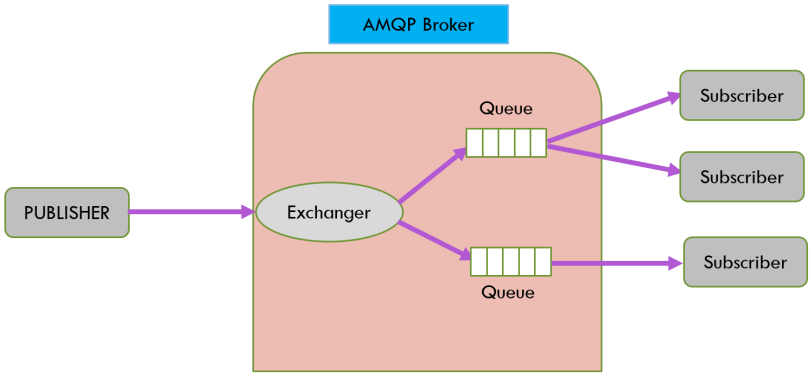

Figure 2.11: AMQP protocol architecture [35].

Although for majority of IoT environments reliability is not a deciding feature for the communication system, there are some few cases where that is indispensable. For instance, in a situation where a luminance sensor that sends periodic data of the measured values, a lost message will not be a huge problem because shortly after, another value will be sent and the prior loss will have almost no impact to the system. However, in a hospital, if an emergency message in not delivered successfully could mean a life threatening situation for a patient. Situations like this ensure the importance of a communication protocol like AMQP in the IoT context.

2.2.3.4 XMPP

Extensible Messaging and Presence Protocol (XMPP) is, like the former, a standardized message-oriented communication protocol and it was created by Jabber open-source community in 1999 for establish near real-time instant messaging (IM), presence information, and others [35].

XMPP, as shown in Figure 2.12 uses a decentralized architecture where there can be more than one server and the messages can be transferred between them until they reach the desired destination, ensuring that way a high scalability [15].

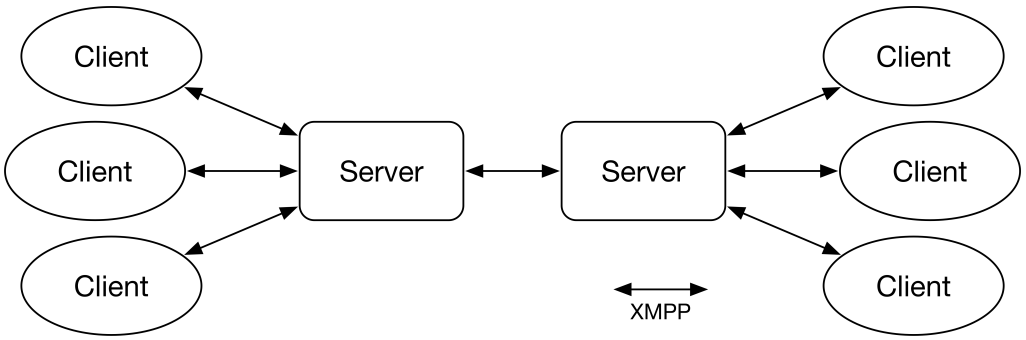

Figure 2.12: XMPP protocol architecture.

Unlike the protocols addressed previously, XMPP supports not only a publish/subscribe architecture, but also a request/response one. XMPP uses an eXtensible Markup Language (XML)-based approach to messaging, which allows multiple extensions of the protocol to be created, making it a highly extensible protocol. However, being that XML-based messages are text based, the communication overhead is higher [33].

Although XMPP was not designed for IoT environments, it still has capacity to be an important protocol in some specific IoT use cases, due to the near real time message exchanging and the high extensibility offered.

2.2.3.5 APPLICATION PROTOCOLS COMPARISON

In this section, the protocols previously addressed will be compared and an evaluation of the environments where each one of them is more suitable is provided. Bellow, in Table 2.2, the protocols are compared based on the type of transport used, the presence of security mechanisms and QoS, and also the support of Publish/Subscribe or Request/Response patterns.

Regarding MQTT and AMQP, both protocols use a publish/subscribe pattern but, although AMQP offers a wider variety of features, MQTT gains for being more lightweight, less complex and easier to deploy in most cases, client and broker wise. For this fact, MQTT is usually the choice for most systems, where there are not very restrictive requirements. Nevertheless, both protocols are suited for IoT environments with constrained and low-power devices.

	MQTT	**CoAP**	**AMQP**	**XMPP**
Transport	TCP	UDP	TCP	TCP
Security	SSL and User/Password auth	DTLS	SSL	SSL
QoS	Yes	Yes	Yes	No
Publish/Subscribe	Yes	No	Yes	Yes
Request/Response	No	Yes	No	Yes

Table 2.2: IoT Application Protocols Comparison.

Looking at CoAP, the greater difference between the last two is the use request/response pattern, instead of a publish/subscribe one. Also, CoAP uses a one-to-one transfer protocol between a client and a server, which makes it more suitable as state change model, instead of an event driven one.

Finally, the XMPP although being a near real time communication protocol with an high number of features, such as allowing both publish/subscribe and request/response patterns, still lacks by not being as lightweight as the other protocols. Nevertheless, this protocol has a place in the market for solutions where reliability is a key factor.

2.2.4 IoT enabled BAS solutions

According to the author of [36], the old industry of BASs systems is being disrupted by the Internet of Things. Only some of the most known BAS companies are trying to integrate IoT principles and open protocols in their solutions in order to keep themselves in a leading market position. The emerge of cheap and small devices with wireless connectivity using open standards and the possibility of creating affordable automation environments, using open-source software, are now in reach, providing a wider variety of solutions to empower buildings with automation capabilities and thus, reduce the energy consumption while increasing the comfort in buildings.

A practical example of a building automation system, using IoT principles and standards, is HCL Technologies BAS. HCL Technologies is a member of the Intel Internet of Things Solutions Alliance [37] and uses an Intel-based gateway to deliver a powerful, yet cost-effective building management solution [38]. In this system, as shown in Figure 2.13, is used an Intel's IoT gateway empowered with HCL BAS software is used, which is responsible for providing a web interface and means for simple automation mechanisms, e.g. turning the Heating Ventilation and Air Conditioning (HVAC) system on when a temperature threshold is reached. The HCL BAS software and gateway supports protocols like ZigBee and Wi-Fi, and the translation between them is made without need of configuration by the building manager.

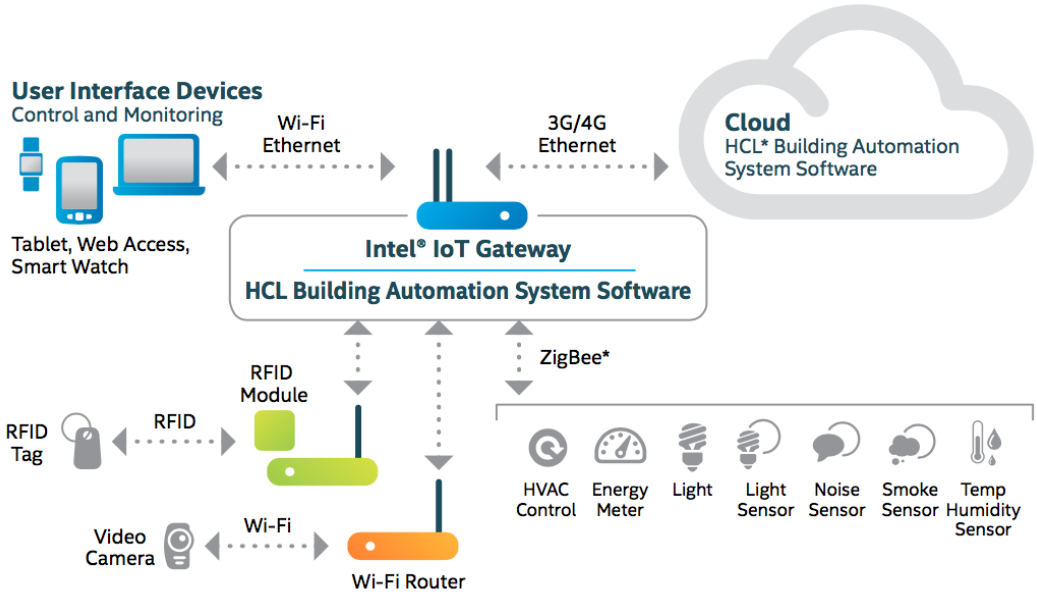

Figure 2.13: HCL Technologies BAS deployment architecture [38].

Although the presented solution constitutes a robust and novel solution as a BAS, with basic automation that can address most of building manager's needs, there is a lack of complex automatic correlations between events generated by the sensors network. In the next section the importance of Complex Event Processing (CEP) modules to enhance the performance of event processing as well as the intelligence of a building through complex correlations is presented.

2.3 FAULT TOLERANCE IN COMPONENT-BASED SYSTEMS

Nowadays, most system architectures are composed by individual components (physical or logical) with well-defined communication interfaces. This can bring several advantages such as the reusability or replaceability of components without affecting the overall operation of the system [39].

Nevertheless, this type of architecture has to be tolerant to failures of components(or nodes), in order to achieve an high availability solution, which is a crucial feature to the majority of systems [40]. As the author of [41] suggests, an error is the manifestation of a fault in the system, and a failure is the manifestation of an error on the provided service by the system. The same author also classifies faults on the base of their sources, manifestations and persistence, namely software errors, materials errors or transmission errors. Therefore, a node failure can be caused either by an internal error

on the software, a physical malfunctioning on the component or the shutdown of the communication links to the node.

In order to have a fault tolerant system, fault detection and fault recovery techniques must be implemented. Bellow, some of those techniques are presented.

Fault Detection Techniques

Fault detection is the process of collecting information about the system's nodes state. This information can be gathered by failure detectors using two different keep alive messages: i) ping; and ii) heartbeat [41]. In the first one the failure detector sends a ping message to the monitored nodes and waits to receive an acknowledge message (ack). If the monitored node is down, due to a failure, the failure detector will not receive the ack and therefore conclude that the node is down. In contrast, for the heartbeat message it is the monitored node that sends a periodic message to the failure detector, informing that it is still running as expected. If the detector does not receive a heartbeat message from a node, after a timeout, it concludes that the node is no longer available.

The author of [41], claims that heartbeat messages have more advantages when compared to ping messages, such as the heartbeat need for half the messages used by ping detectors. However, the ping detectors only have to preform time control in case of a failure in receiving an acknowledge, while the heartbeat detectors need to keep track of the time to detect the absence of heartbeats after a timeout.

Regarding the fault detector's location in the system, the author of [42] identifies two approaches, a centralized or distributed detector. The centralized approach uses a single node to perform the monitoring of other components, however, although this approach being efficient and accurate to identify faults [42], it constitutes a single-point of failure in the system. Plus, in large-scale systems, the traffic generated by these messages can congest the network, resulting in a more expensive and inefficient solution. The distributed approach uses several nodes as failure detectors, which eliminates the single-point of failure problem, however, introduces more complexity to the system [41].

Fault Recovery Techniques

The fault recovery occurs immediately after a detector senses the fault of a system's node. According to the author of [43], the high availability of a system is normally achieved using redundant entities, whereby, when a component fails, the redundant node takes over its tasks. The author of [44] states that there are two types of physical redundancy: i) active replication; and ii) primary backup.

Regarding the active replication, it is usually addressed as a N-modular redundancy approach [43], in which a set of replicated modules receive the same input, and the resulting outputs from the different modules are compared and the result with more concordance(majority) is elected as the output. In Figure 2.14 this active replication method is depicted. The most common N-modular redundancy is the Triple Modular Redundancy (TMR). TMR uses a set of three replicated modules, thus, it allows one module to fail, or produce a faulty value, whilst the remainder correctly operating modules are still able to mask that failure.

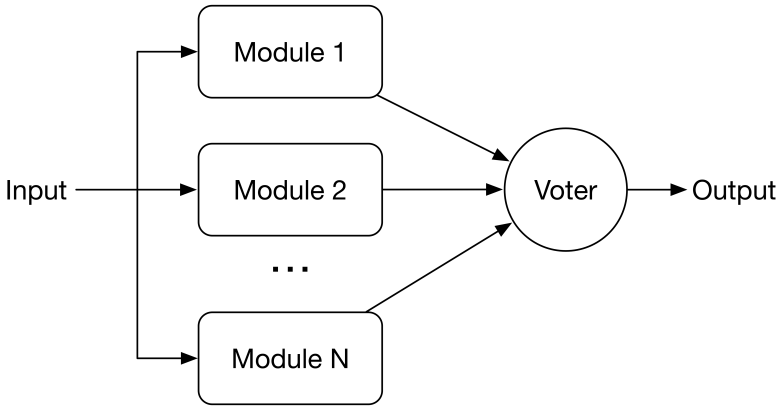

Figure 2.14: N Modular Redundancy.

Concerning the primary backup approach, also known as hot standby redundancy [43], it uses a primary node, and one or more unused backup nodes. In this approach, there is a need for a failure detector so that when the primary node fails, the backup node can be notified to take over the processing. In Figure 2.15 the hot standby redundancy is represented, and as can be observed, both the active and the passive replicated module need to be connected in order to be synchronized [43].

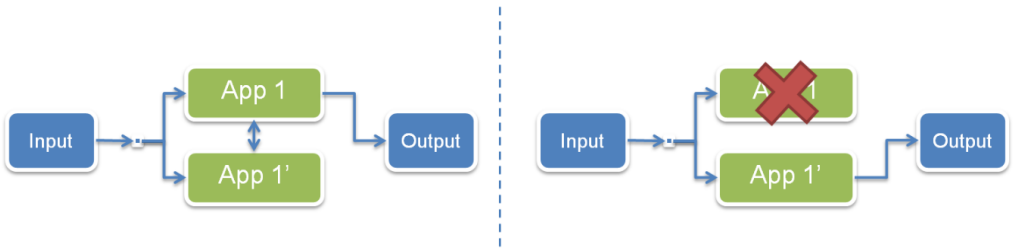

Figure 2.15: Hot standby redundancy. Adapted from [43].

The author of [43] also states, as an example, a situation with three hosts connected through a common communication bus. Each host need to send periodic heartbeat

messages to this bus and, if a host's heartbeat is missing, due to a failure in the host or in the network, the other hosts can act immediately to resolve the failure.

The methods presented in this chapter can be used in any component-based system to achieve an acceptable level of fault tolerance, however, each method can be more suitable to different situations.

2.3.1 FAULT-TOLERANCE IN IOT

In typical smart environments, sensors and actuators are connected to a gateway, also called a sink, that is capable of communication and processing the received data, either by itself or by sending to another component responsible for doing so [45]. Therefore, three core components of a smart environment that have a direct effect on the whole system operation, can be considered: devices, gateways and automation engines. A device malfunction can only be surpassed by human intervention when it has no self- healing mechanisms. Otherwise there would have to be redundant devices. At the gateway level, since it is a sink device where the information flows to, redundant gateways would also be necessary. Otherwise, in case of multiple gateways connected to different devices, the other gateways should be able to sense that there was a disconnected gateway and takeover the control for the lost devices. Lastly, the automation engines, like the other components, could resolve a failure situation by having redundancy or self-healing techniques implemented in the application. However, in case of irreversible failure, gateways should be able to sense that the automation components are unreachable and, in a perfect scenario, having automation logic implemented so that the system could work at an emergency state until the automation engines are re-established.

2.4 AUTOMATION LOGIC

Automation can be described as actions preformed without human intervention. In the present dissertation context, automation is achieved using a logic component that receives all events, gathered by sensors from the environment, and infer actions based on buildings manager preloaded rules. For instance, events like movement in a room or temperature and humidity changes, are sent to this automation component that, based on a set of rules, can send triggers to turn on/off lights, HVAC units or others.

In a small house, the number of sensors and events are fairly small and thus, there is no need for a powerful and complex automation unit. However, in a building context,

the amount of events generated is much higher, which requires the use of a more efficient and high performance platform. Also in an environment with so many variables, events cannot be treated isolated from the greater context, so there is a need for a platform that able to correlate these isolated events and infer complex action-reaction relationships.

In the next subsection, a system capable of inferring actions based on processing and correlating large event streams is presented.

2.4.1 COMPLEX EVENT PROCESSING

With the emergence of IoT in different areas, huge amounts of data, that need proper analysis, are produced. However, in areas like smart homes and buildings, the information streams need to be processed almost instantly in order to infer quick actions. CEP is a technology used to process and analyse large streams of data, from multiple sources, and infer real-time conclusions or actions based on the detection of patterns and complex events. As illustrated in Figure 2.16, a complex event is inferred when a certain number of isolated events happen in a specific pattern.

This processing methodology is used in different contexts where a real-time and high performance platform to process data is needed. One of these cases is the financial industry where, in order to take low risk and profitable actions, CEP is used to deduce real-time trading predictions.

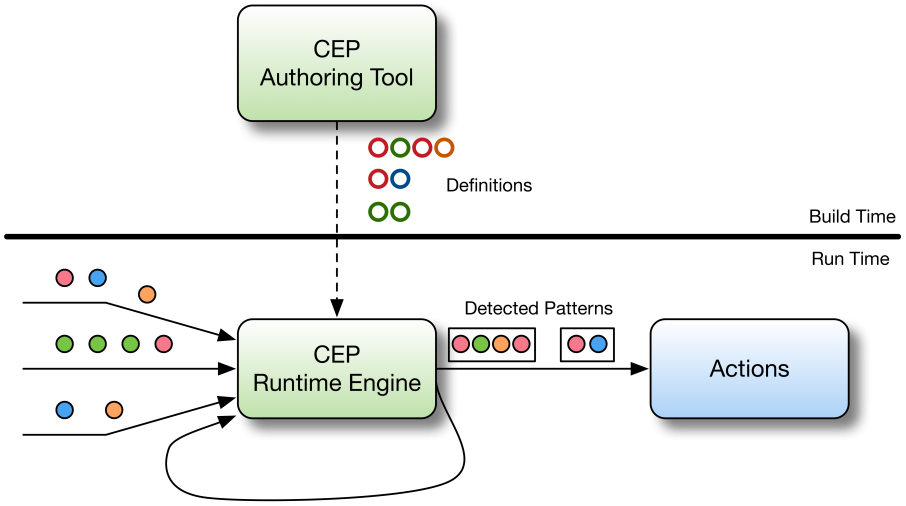

Figure 2.16: Detection of complex events using Complex Event Processing.

As stated before, smart buildings could also use CEP to greatly increase the quality of its automation. In particular, with a CEP engine, the events generated by the

network of sensors can be analysed to derive actions that, with a classic automation system, would be impossible. As a simple example, in an empty room where the air conditioner is turned off, a sharp decrease of temperature over a small period of time could mean that a window was left open and thus, a message could be sent to the building manager alerting this situation. In this example, with the gathered data from different sensors, such as temperature and presence, over some time, the Complex Event Processing (CEP) engine was able to infer a conclusion. This type of conclusions are only achieved if the automation system takes into consideration multiple events across time instead of analyzing them separately.

A CEP engine implements a set of tools such as filtering, time windowing, aggregation or pattern detection of events, and makes possible to design complex rules using these mechanisms. They can be implemented either by using a query-based model using a query language similar to SQL, or a rule-based model, where a rule is divided in three components: an event, the condition that is evaluated when the event arrives and the action that is taken when the condition is satisfied.

CHAPTER 3

Smart Building Solution

In this chapter it is provided a solution to improve efficiency and usability of a building by adopting automation mechanisms which enhance the intelligence of the building. Considering IoT principles, this system should be capable of processing large data streams in real time, analyzing and correlating events, and react and adapt to emergency states and always guarantee a minimum functionality. This solution is intended to integrate the SmartLightinhg project which is addressed in this chapter.

This chapter begins with a brief description, in section 3.1, of SmartLighting project and its intentions of developing a smart environment of automation and control in IT2 building. Later, to ensure that all system needs are addressed and taken into account, in section 3.3 a survey of necessary capabilities and requirements to satisfy stakeholders is done.

Since the main aim of this dissertation is the failure handling and adaptation of the gateways to emergency states, in section 3.4, five critical scenarios of operation are addressed.

Finally, in section 3.2, a presentation and evaluation of the system architecture, and its components' interactions are discussed.

3.1 SMARTLIGHTING PROJECT

3.1.1 PROJECT OVERVIEW AND OBJECTIVES

The SmartLighting project[46] aims to create a smart environment in IT2 building in order to automate lighting and HVAC infrastructures. IT2 building has been running with Compact Fluorescent (CFL) luminaries, which, are often not only, inefficient and hazardous for the environment, but also, in most cases, unable to support variation of the luminous output, i.e. to be dimmed.

This project intends to remove all these CFL luminaries by Light-Emitting Diode (LED) luminaries, that can resolve all previously stated issues and also be combined with sensors. These can used to collect data from the environment such as luminance, temperature, humidity, motion and others. The collected data is then used to act in real-time to the changes in sensor values.

Such system needs to be backed up by an intelligent management platform in order to react based on a set of rules and input streams of sensor values. Moreover, there will be mechanisms to correlate events and determine patterns that enable a more immediate reaction. As an example, if a user has the habit to turn on the AC every day in the morning, the system will be able to learn and automatically replicate that action as soon as the user arrives in the building.

Also, the system can use external sources like meteorological forecasts to take preventive actions, such as increasing the dimming levels of luminaries and the building temperature in cold rainy days.

Additionally, users should be able to set their own rules to enhance the comfort in their offices. This could be done either by using a mobile application or a web interface. Also, these user platforms can enable use of notifications, providing ways to alert the building occupants of important events. For instance, users can be informed about future meetings, the presence of another occupant in the building, and more.

Finally, an important feature for this project is the failsafe mechanisms to ensure that all basic automation needed for the building's proper operation. This includes such things as the trigger of a motion sensor to light up corresponding luminaries, which should work even in situations of failure in core elements of the architecture or network communications. This can be achieved by giving gateways, the component that communicates directly with the sensors and actuators, lightweight processing mechanisms to ensure the basic functionalities described before. This will be the main focus of the presented dissertation.

3.1.2 STAKEHOLDERS

The success of a project depends directly in the people involved, either by developing its features or simply by consuming them in its final form. A correct identification of the stakeholders, their motivations and expectations, plays an important role in the success of any project. In SmartLighting project, 4 stakeholders were identified, namely:

In SmartLighting project, 4 stakeholders were identified, namely:

- Building occupants
- Building owners
- Building managers
- Developers team

The primary stakeholders in this project, are the IT2 building occupants and owners. While occupants will take direct advantage from the implemented features due to the enhanced comfort and usability of the building, owners gains will be economic as far as the building energy consumption and maintenance costs will be reduced.

As for building managers, the benefits come from an easier interaction and configuration of the whole system. A more precise aware of system's components status and a notification system, can help prevent or quickly react to systems failures.

Lastly, the developers team take a huge benefit in the success of the project since they are interested and devoted to create the best product possible that fulfils all the other stakeholders expectations.

3.1.3 USE CASES

Based on the stakeholders' survey done in the previous section, the two main actors for this BAS solution are the building's manager and the occupants. Those actors directly use the platforms and resources provided, to enhance the comfort of the environment around them as they please, so the main focus for user driven cases will be in those two.

Although there are many use cases for both the building's manager and the occupants, in this section will only be addressed those that were already implemented in this project, giving a greater understanding of the overall project's functionalities. Later, in section 3.4, the use cases and scenarios introduced by this dissertation work are presented.

Building occupant

The building occupants will be the principal users to interact with the system. This interaction can be achieved by using either a mobile application or a web interface. The use cases addressed bellow combine the most common and frequent operations:

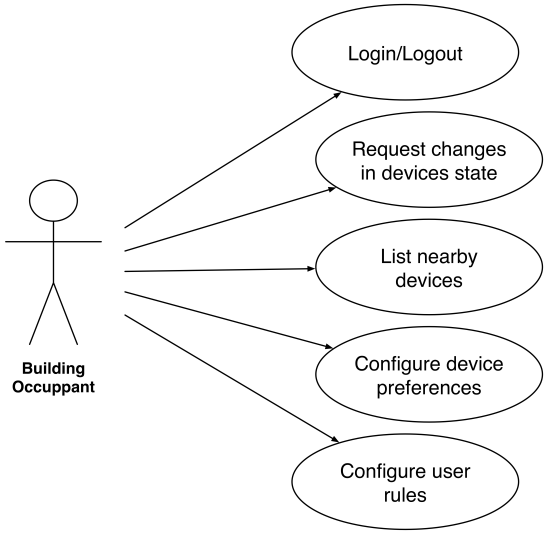

Figure 3.1: Use case diagram of building occupants interaction with the system.

- **Login/Logout:** This use case is intended to enable the access to user oriented features. The building occupants can login to the platform, either using the mobile application or the web interface, to access preferences and personal environment control features.
- **Request changes in devices state:** The building occupant can request a change in the state of a device, that can be accepted or denied by the system depending on the access rights of the user. For instance, an occupant can request a change in nearby devices like air conditioners, lights, and others.
- **List nearby devices:** This use case allows users to request the list of nearby devices, based on its location, allowing to access or request changes to devices in the same room as the occupant, giving that these are the most likely to be needed access.
- **Configure device preferences:** The user can create or edit device pre-sets, allowing the selected values to be applied automatically to the device in future uses. For instance, the air conditioner in a user office can be pre-set to 21ºC so that when this device is turned on, that value is automatically set. These changes would also be checked against the user access rights.
- **Configure user rules:** This use case allows building occupants to create rules that perform a set of actions automatically, giving the right conditions. For instance, the user could create a rule to automatically turn the air conditioner on if the temperature in his office exceed a giving value.

Building manager

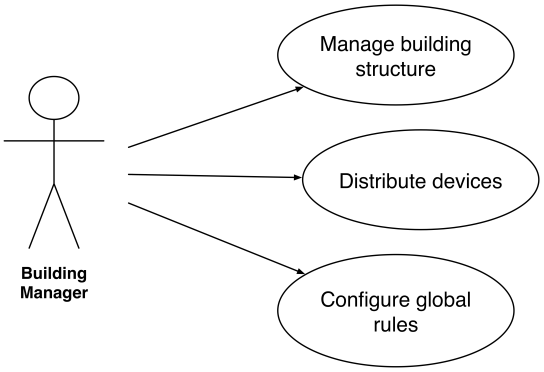

Figure 3.2: Use case diagram of building manager interaction with the system.

- **Manage building structure:** The manager can create the representation of the building in the structure management platform. This allows the creation of a virtual portrait of the floors, rooms and areas of the building.
- **Distribute devices:** This use case permits connected devices to be distributed through the areas and rooms of the building's virtual representation.
- **Configure global rules:** The building manager has the ability to add, delete, enable or disable rules for the whole building. These core rules' main purpose are the increase in building´s efficiency. For instance, instead of leaving the corridors lights always on, a rule can be created to turn them off when there's no one walking by. There is also a mechanism to test the rule before the deployment.

3.2 SYSTEM ARCHITECTURE

Taking into account the objectives for this dissertation, and the existing architecture for the SmartLighting project, the proposed architecture has in mind, not only IoT and complex event processing principles, but also the referred features for this dissertation such as failure handling and rules distribution through gateways.

In this section, the global system architecture for this project, represented in Figure 3.3, is presented alongside with a description of all the components that compose the solution.

Note that components like the CEP engine, and the Building Manager (BM) are already implemented in a past dissertation of the SmartLighting project [2].

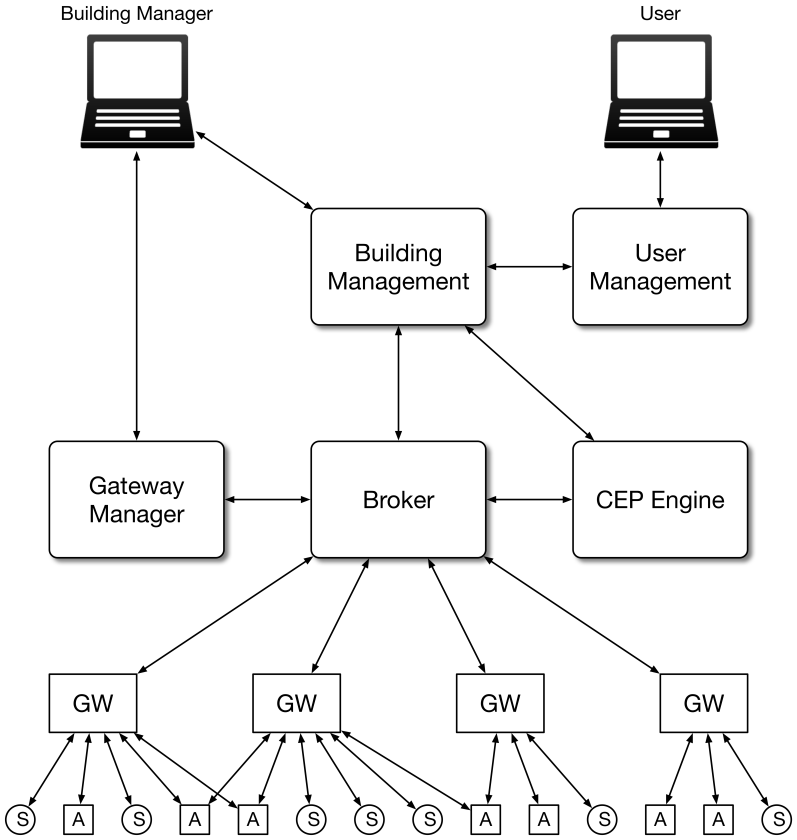

Figure 3.3: Smart Building: Architecture diagram

3.2.1 COMPONENTS OVERVIEW

User Management

The user management platform is responsible for giving to common users of the building, the ability to interact with the platform through a web page. Users can manage their own building rules and control field devices directly. This can be done by publishing on the broker the desired changes, or by communication directly with the CEP Engine or the building management platform.

Building Management

The building management component is a user friendly platform that aims to facilitate the rule management and the configuration of the building's schema. The building manager can create, edit or delete rules in a simple intuitive way and this component later translates them in a complex language that the CEP Engine can understand. Also, through a virtual representation of the building, the BM can

distribute devices across all its areas. After the correct configuration of the desired devices, this component is responsible for providing the information of how should the CEP engine, and the other blocks of the architecture, address to this device in order to send an order to change state or report new sensor data to the CEP Engine.

CEP Engine

The CEP Engine is a crucial component in the proposed building automation solution. This engine must be a real-time and high-performance component, able to process hundreds of sensor triggered events, per second, based on complex rules created by users. The actions performed by this engine can either be directed to an actuator or to a component in the platform. For instance, sending a notification to the building manager when there are irregular sensor readings.

Broker

The broker component, which refers to a message broker, is responsible for receiving and routing information across the connected components. Every component connects to receive and send information, to and from other components.

Gateway Manager

The gateway manager is the component responsible for rules distribution to gateways and fail handling. The first one, since gateways will be equipped with the ability to process events, in the case of a failure in the CEP Engine, there is the need for a regulator to distribute and be always aware where the gateways' rules are being deployed. In case of a gateway failure, it is also responsible for distributing the devices and rules from that gateway, to other ones capable of supporting them. Also, the gateway manager should guarantee a balanced number of rules and devices across all gateways. For instance, if two gateways are in the reach of the same device, the gateway manager should allocate that device to the one with fewer devices.

Gateways

Gateways are responsible for discovering devices and communicating with them. These nodes have to convert sensor events into understandable messages to the automation engine, and vice versa. Also, gateways should be equipped with a lightweight automation engine so that, in case of a failure in the CEP engine, or in the broker, the gateways can continue to process events, and thus, a minimum downtime in the system's operation.

Sensors and actuators

The field layer represents the set of devices, both sensors and actuators, used to gather information and interact with the environment. The information gathering and detection of changes is done by sensors, that send this information to the respective gateways in the network layer. After processing of this data, either in the aggregation and automation layer, or even by gateways in the network layer, the orders for changes are sent to the actuators.

3.3 REQUIREMENTS

The proposed solution must be designed to be integrated in a real-world scenario. Accordingly, the requirements, in this section, must converge and be in conformity with the project's objectives stated in section 3.1.

Fast Responsiveness

The system's performance and fast responsiveness are huge requirements for a successful BAS solution. The system must react to events with such responsiveness that the occupants experience and comfort is not affected. For instance, when a user enters a room, it is expected that the lights turn on in a question of milliseconds providing an instantaneous feeling to the user. In other cases, a slower responsiveness is not critical for the occupant. As an example, in the heating system, when the temperature goes below a certain threshold, if the air conditioners take a few seconds to turn on, it won't be noticeable for the users.

In addiction, the system's management platform responsiveness is another performance requirement as it shouldn't feel slow to the user.

Flexibility

One important requirement for this project is flexibility. Firstly, in order to have a system that can be easily upgradable, components must be independent of hardware. For instance, the gateway software must be developed in such way that can work in a large number of micro-controllers and low power System on a Chip (SoC) computer. Also, the communication protocols between system components should be able to be changed allowing to meet different requirements of different scenarios.

Finally, the system should facilitate the integration of different type of sensors or actuators, without the need verbose and difficult configurations.

Scalability

The system should assure that with the addition of devices, the overall performance does not decline. Also, it should facilitate the horizontal scalability of system nodes, making it transparent and easy to integrate new servers to maintain or improve system performance.

Failure Handling and Basic Functionalities

Failure handling and the assurance of basic functionalities were the main goals for this dissertation, making this requirements of the utmost importance.

The whole system is designed to fulfil a set of expectations and to work perfectly at any given time, however, there could be some cases where due to a critical system failure, either a downtime in the CEP engine, communication errors or a congested network, the whole system could start malfunctioning. In more severe cases this could mean, for instance, an entire blackout in a building, or a fire sensor not triggering the alarm leading to serious injuries to the occupants.

With all this facts in consideration, a significant requirement for a BAS solution is the ability to always be aware of the entire system state and to react immediately to abnormal system occurrences. This means, for instance, that in case of communication failure between the sensors and the CEP engine, the basic functionalities like lights turning on or off, alarms, doors and other key functionalities still work autonomously.

Data Correlation

In order to do a more accurate and elaborate reaction to sensing events, the event processing unit should be able to correlate events and infer decisions based on previous learning. For instance, if a user always turn on the air conditioner, when the temperature is bellow a certain threshold, the system should be able to learn from this behaviour and promptly react without human intervention.

User Control and User-friendly Interfaces

The system should allow users, based on their roles and permissions, to manually alter the state of a device. This means that interfaces, web or mobile, should be available to the users to control the surroundings. These interfaces should be implemented, taking in account that not all users have the expertise to interact with verbose configuration parameters, thus, the interfaces should be as intuitive and perceptible as possible.

3.4 OPERATIONAL SCENARIOS

In order to achieve an environment where the minimum functionality of the system is always assured, for example, the lights turning on when someone walks in a corridor, there can be no system component that, in case of failure, could put in risk those functionalities. With this in mind, and looking to the existing architecture, simplified in Figure 3.4 only with components that directly and actively affect the event processing capability, five operational scenarios can be identified.

Scenario 1 - All components are operational

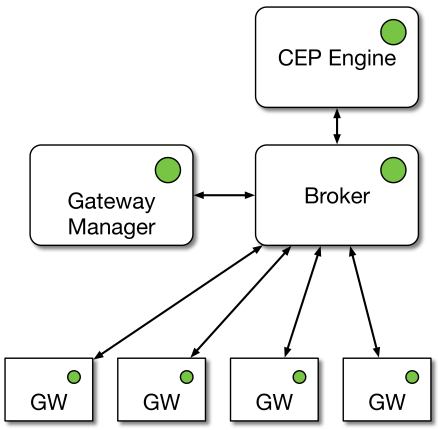

Figure 3.4: Simplified SmartLighting architecture.

This first scenario corresponds to the normal operation of the system, as shown in Figure 3.4, where every component is working properly. In this case, the CEP module should receive events, generated by devices and sent to the message broker by gateways, process them and send the result to the message broker, and later to the corresponding gateway.

Scenario 2 - CEP Engine failure

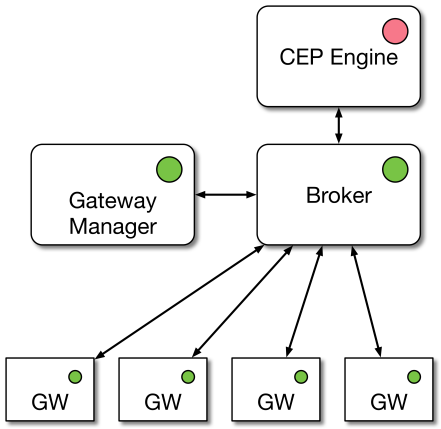

Figure 3.5: Functioning scenario 2.

In this second scenario, shown in Figure 3.5, the CEP module fails and thus, the event processing is halted, either because the server where the program was running was down, or because of a failure in the connection between this component and the message broker. Either way, in this scenario, the generated events cannot reach the CEP Engine to be processed. Since one of the requirements for this project is granting that the building's basic functionalities, like lighting, still work, the system should be able to detect this failure and gateways should begin to process events.

Scenario 3 - Gateway Manager failure

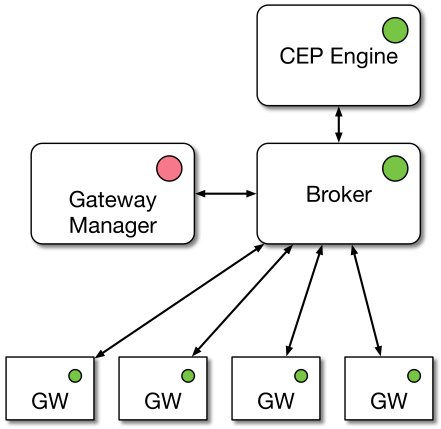

Figure 3.6: Functioning scenario 3.

Scenario 3 represents a failure in the Gateway Manager while all the other components are working correctly. Again, this failure can be due to a program error, a downtime in the server where the program is running, or because of a failure in the connection between

this component and the message broker. When one of these situations happens, the system should continue to work correctly, however, since this component is responsible for distributing rules and devices through the accessible gateways, this features will not be available and thus, the automation of some building areas might stop working.

Scenario 4 - Gateway Manager and CEP Engine failure

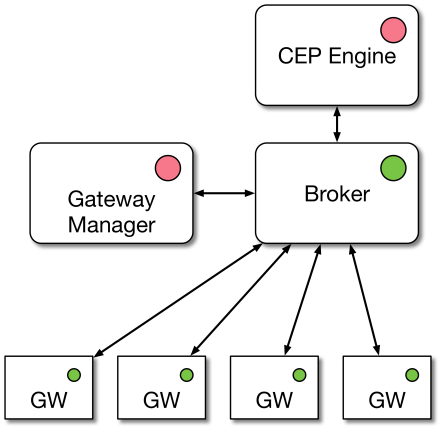

Figure 3.7: Functioning scenario 4.

This scenario corresponds to both the Gateway Manager and the CEP Engine being unavailable. This scenario is a combination of the failures shown in scenario 2 and 3, as both the CEP Engine and the Gateway Manager are no longer available. Again, this can happen if the two components fail internally at the same time, or the communication between them and the message broker is down. Based on what each component is responsible for, this scenario would made unavailable the processing of events in the CEP Engine and the rules and devices distribution to gateways performed by the Gateway Manager. In this scenario the system should continue the processing of events in the gateways, and make use, if necessary from the message broker.

Scenario 5 - MQTT Broker failure

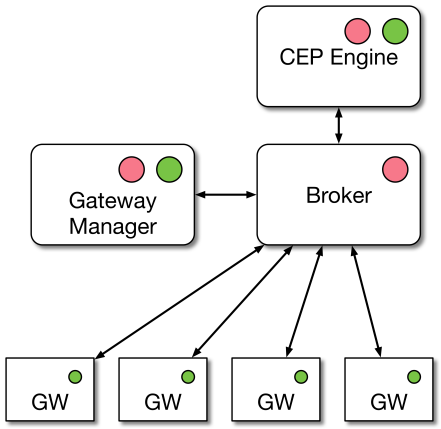

Figure 3.8: Functioning scenario 5.

The third scenario, shown in Figure 3.8, when the message broker is down, the communication between gateways and the CEP Engine is cut off. Even if the CEP Engine and the Gateway Manager were still running, they would not be able to communicate with the gateways. This scenario can happen due to a network failure in the building. In this case, since every component, except the devices and gateways, are running in building servers, if the internal network was down, the gateways would be isolated and the devices would only be able to rely on the respective gateways, since the connection between the two would not be affected. This would be the most severe scenario, and the system should be able to grant the minimum functionality by, for instance, process the device's events on the gateways, as said before.

In every scenario stated before there is a need for failure recovery of the gateways too. For instance, if a gateway gets disconnected, the other gateways should be able to take over the sensors and actuators previously owned by the failing gateway and act as a new gateway for them.

CHAPTER 4

Implementation

The objective for this chapter is to describe the steps and decisions taken, and the reasoning behind the implemented features. It starts in section 4.1, by presenting the objectives for the implementation, following by the used technologies on the several components in section 4.2. After that, in sections 4.3 and 4.4, it is presented the standards used to access the devices and to define rules, respectively. Following, in section 4.5, the whole system's architecture, as well as the functionalities of its components, are described.

Since one of the requirements and objectives for this project was the existence of failure handling mechanisms, in section 4.6, it is described the implementation taking into account the several scenarios addressed in the previous chapter.

4.1 OBJECTIVES

Although the implementation of the whole solution, presented in chapter 3, would be optimal to validate the project, it would be unrealistic due to the amount of time necessary to do so and the limited time available for this dissertation. Since some of the features were already implemented, such as a working platform to create and manage rules, a CEP engine and a simple gateway to communicate with devices through BLE, this dissertation's main focus was in the implementation of gateways capable of implement automation rules, alike to the ones implemented by the CEP engine, as well as give them means to adapt to failing and emergency states. For that reason, was also implemented a gateway manager to act as a monitor for the gateways. The gateway manager is not only able to control the operation of each gateway, but also, manage the distribution of rules and devices throughout them.

Looking at the architecture presented in chapter 3, the component for the user management dashboard was not implemented for the scope of this work. Apart from this, all the features mentioned were implemented and are detailed in the continuation of this chapter.

4.2 ADOPTED TECHNOLOGIES

This section aims to discuss the technologies chosen for the different components of the architecture. These choices will be justified and explained taking into account the requirements for this project. However, it is important to note that there are no perfect solutions and, in some cases, several approaches could have been taken.

Since there was a dissertation[2] that already implemented some features like the CEP engine and the BM, some of the technologies used in this dissertation were influenced by past decisions. As far as the CEP engine concerns, the choice was the WSO2 CEP [47], since its features addressed all requirements, that such system would need, to fit this project's requirements. For the purpose of this dissertation both the CEP engine and the BM as well as the communication between them, did not affect any of the new implemented features. Since the MQTT protocol was primarily used as the elect protocol for communications in the referred dissertation, the implemented features presented in this work followed the same path.

Finally, it is important to state that the standard used, in SmartLighting project, for both object definition to read and write information to/from devices, and the rules structure, were also maintained and will be described in the following sections.

4.3 ACCESS TO DEVICES

Devices like sensors, which measure environment changes, and actuators, that may trigger mechanisms to make changes in that same environment, are fundamental pieces to implement a smart and automated environment. Since there are a huge variety of devices with different characteristics, in order to enable extensibility for the system, the access to those devices, as well as the messages format, should be standardised. In another dissertation for this project, a standard following the IP Smart Objects Alliance Guideline[48] was made and is illustrated in Figure 4.1

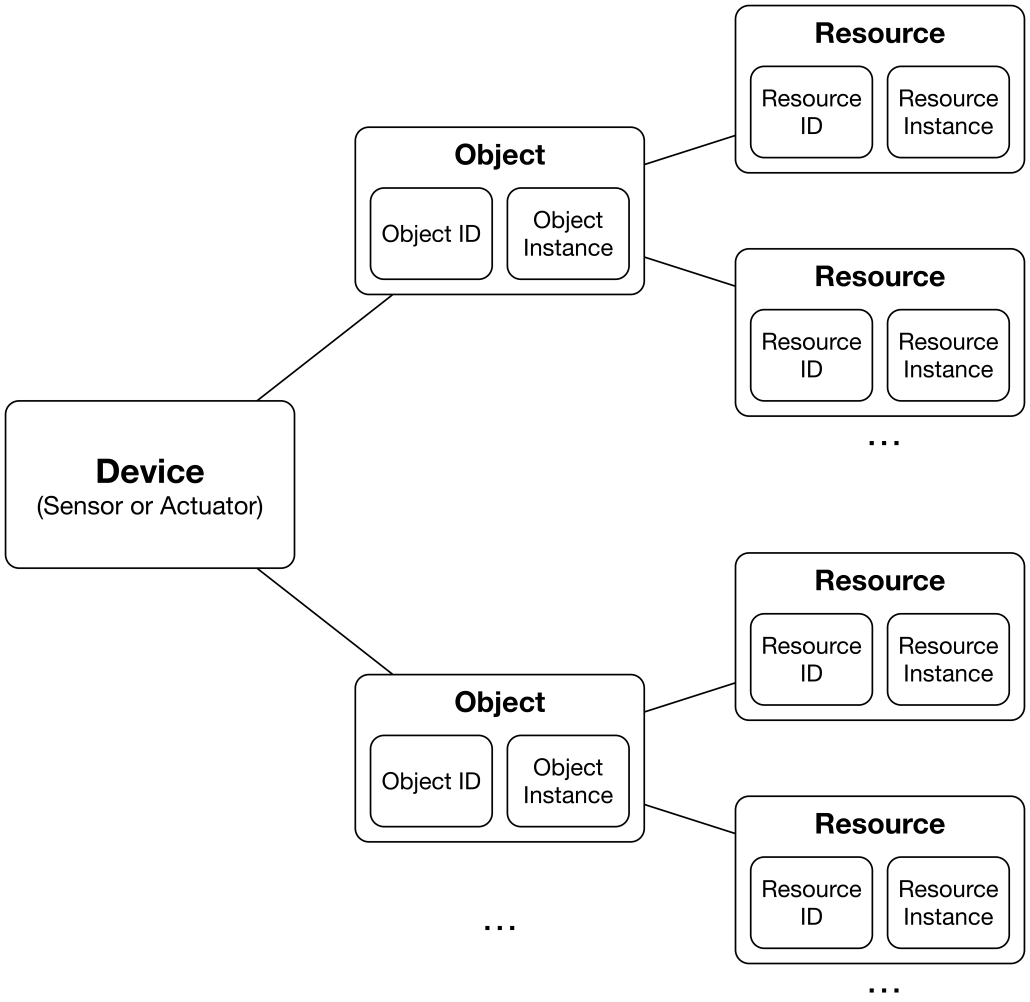

Figure 4.1: Device objects representation

In this representation, a device can have multiple objects (sensors and/or actuators), identified through an ID and an object instance. As example, a device can have multiple sensing or actuating capabilities, each one of them identified with a different ID. A device with two motion sensors, will have two objects with the same ID but different

object instances.

Inside the representation of each object, the information is divided in resources. Resources represent the available features that each object provides, such as the available values that can be read or written, or the actions that can be triggered. As an example, an luminaire actuator object can have resources to read the current state of the luminaire, turn on/off or dim the light to a certain value, each one of them represented with a different ID. The resource instance is used when, per example, an actuator for aluminaire with two lamps offers the same resources for each lamp separately.

Using this representation, the properties of a device can be accessed using an Uniform Resource Identifier (URI) the following way:

```
.../Object_ID/Object_Instance/Resource_ID/Resource_Instance
```

For instance, a device with a temperature sensor that offers the possibility to check the current state or check the minimum value measured by the sensor since it is ON, can be illustrated and accessed the following way:

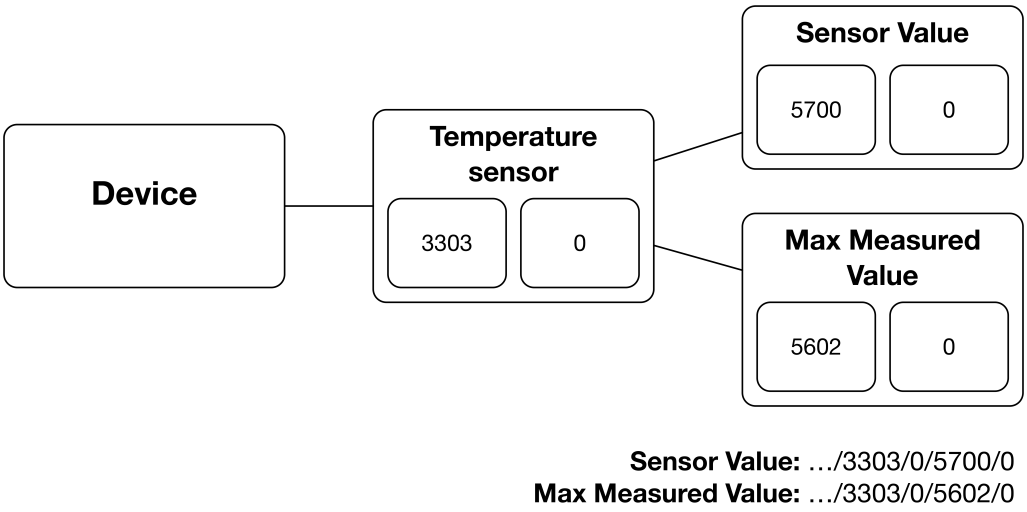

Figure 4.2: Access to a device resources.

Since the protocol used for this project was MQTT, which allows to subscribe and publish messages in specific topics, this URI can be used in the topic to address each device resource.

Since the WSO2 CEP was chosen as the complex event processor for this project[2], a message format to communicate with devices, supported by that engine, was chosen. The format is represented in Snippet 1.

```
{
    "event": {
        "metaData": {
            "attribute_1": ***,
            "attribute_2": ***,
            ...
        },
        "correlationData": {
            "attribute_1": ***,
            "attribute_2": ***,
            ...
        },
        "payloadData": {
            "attribute_1": ***,
            "attribute_2": ***,
        }
    }
}
```

Snippet 1: Example of a simplified message for an event sent to a device.

These event messages are divided in three main logical sections: Payload Data, Correlation Data and Meta Data. Payload Data is the most important data to be transported, such as, the values that are sent to and from devices. The Correlation Data transports information that allows correlating events and lastly, the Meta Data is where other attributes that describe the event can be included. In Snippet 2 is represented an example message of an event sent to a device.

```
{
    "event": {
        "metaData": {
            "operation":"set"
        },
        "payloadData": {
            "value": 15
        }
    }
}
```

Snippet 2: Example of a simplified message for an event sent to a device.

This message would set the value 15 on a device's resource, if sent to the corresponding topic using the format presented above.

4.4 RULES STANDARD

To obtain a fully autonomous system, rules must be designed in order to achieve it. In a former work for SmartLighting project[2], a solution was designed to, instead of using directly the languages provided by each CEP engine which are difficult to map using a Graphical User Interface, a more extensible and pluggable solution was created. When a rule is made in the platform, a JSON is generated containing all the information needed to convert the rule to the desired engine, either the WSO2 CEP engine or the Gateway engine, that will be addressed later.

In this solution, a Rule is divided in Actions, each one containing a Target and a Function. The Target specifies to where the output events, resulting from applying the rule, should be sent, while the Listeners inside the Function specify which input events trigger the rule. Still inside the Function, there can be present four types of modules to transform the event stream: Aggregators, Converters, Windows and Filters. Filters are used to select only events that pass a certain condition, otherwise the rule is not applied. Windows, allow to capture events in a given interval of time or occurrences following some criteria. This module is often used with Aggregators that perform aggregate calculations, such as summing the events or calculating averages. Lastly, the function of Converters is to make calculations with each event, for instance to perform unit conversions. There can also be used the Pattern and Sequence modules that, as the name says, are used to detect patterns or specific sequences of events.

In Snippet 3 in Appendix A, is shown an example on an output JSON of a rule with a single action. As can be observed, a rule can be constituted by sub rules. Each sub rule within a Rule has the same Function but different Targets and Listeners. In this particular case, the rule has a function called "setif_value_percent" with two different types of Listeners, a "listen_boolean" and a "listen_value". The first is used to listen to boolean values (which in this case are the motion sensor state) and the last to listen to float values (luminance sensors, in this case). The boolean data is read for a 6 seconds time window and aggregated with the “any” Aggregator which detects if there was any motion in the last 6 seconds. The data containing the luminance values, is aggregated with an average over the last 5 events, and converted using the converter “lux_to_percent”. Therefore, the function sends an event to the target with the 100% calculated percentage, if a motion is detected in the last 6 seconds, or 50% otherwise.

4.5 ARCHITECTURE

Taking into consideration the objectives and adopted technologies addressed in chapters 4.1 and 4.2, the implemented architecture is represented in Figure 4.3. Comparing this architecture to the one presented in Figure 3.3, it is clear the use of MQTT as the main communication protocol, the removal of the User Management component, since it was not in the scope for this dissertation, and, regarding the CEP engine and the Building Manager, those components were already implemented, as stated before. The BLE Adapter which is part of the gateway was also already implemented. Lastly, the MQTT broker used in this implementation was Mosquitto[1], which is a lightweight broker suited for IoT environments, which implements the latest versions on MQTT protocol.

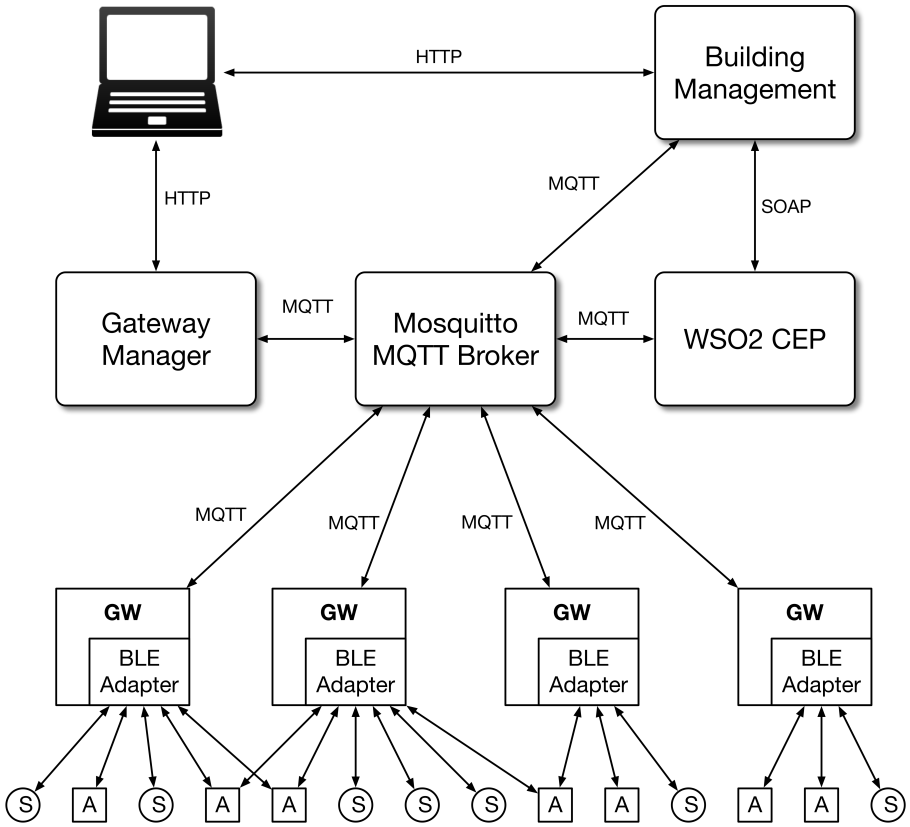

Figure 4.3: Implementation Architecture

For the scope of this dissertation, the focus was the implementation of a Gateway with ability to configure and manage devices, and with a lightweight automation engine that is able to parse rules and process events based on them. It was also implemented

[1]https://mosquitto.org/

a Gateway Manager with features like rule parsing and distribution among gateways and failure handling and synchronization of gateways.

Regarding the failover strategy used, the detection of failures was insured by making every component report its state to the central broker, by publishing a periodic MQTT heart_beat message. This way, if a fail occurs, the other working nodes are able to detect the absence of the failing node and begin recovering from the fault. All the processes regarding these features are explained next.

4.5.1 GATEWAY MANAGER

This section describes how the Gateway Manager fits in the fulfilment of the requirements and objectives proposed for this project, and also how its features were thought and implemented.

This component was programmed using Python 3[2] and since MQTT was the chosen protocol to communicate with other nodes, the Eclipse Paho MQTT Python client[3] library was used. This client enables the application to connect to a MQTT broker in order to publish messages and subscribe to topics.

Rule Parsing and Distribution

One of the Gateway's Manager roles is to receive rules, developed in the existing Building Manager platform. As addressed before, a rule is represented in JSON format, and when the rule is set to be deployed in the gateway's automation engine, the Gateway Manager parses it in order to divide the rule in sub rules and distribute them among the available gateways. To keep track of which rules are deployed where, and to which rule's different targets and listeners correspond, the Gateway Manager platform (GM) attributes a unique ID to each rule and stores, in data structures, the topics(from input and output events) used in each rule and to which rule IDs they correspond, since different rules can have the same targets and/or listeners. These data structures will be important to implement the fail handling measures, which will be addressed later. In Figure 4.4, can be observed the data structure used to store this information. Each entry in the list is a tuple containing a target topic(or listener topic) and a list of rules ID's that contain the same topic.

[2]https://docs.python.org/3/
[3]https://pypi.python.org/pypi/paho-mqtt/1.3.0

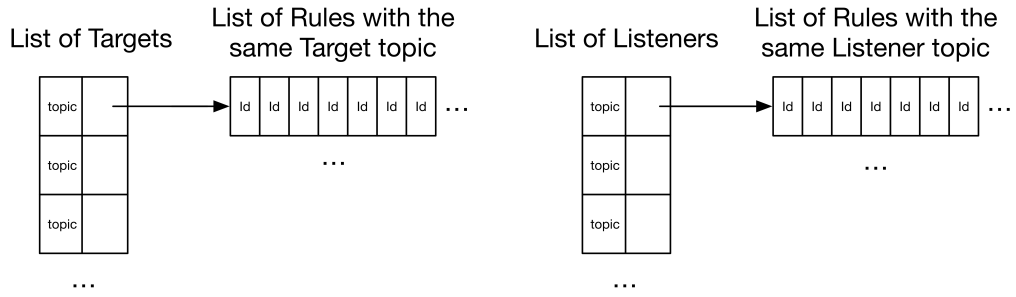

Figure 4.4: Data structures used to store Targets and Listeners and to which Rule IDs they correspond.

After the rule is parsed, the GM distribute the sub rules to the available gateways using a round-robin method, in order to maintain a equal number or deployed rules among gateways. This approach was chosen so that gateways can have a balanced load between them. Other approaches could have been implemented, such as distribute rules by proximity to the devices that they refer to, but as downside, there could be gateways with an extensive number of rules that could put in jeopardy the performance and responsiveness of event processing.

In Figure 4.5 can be observed an interaction diagram of the communication needed to add a new rule to the Gateway Manager and a Subrule to a Gateway, using MQTT.

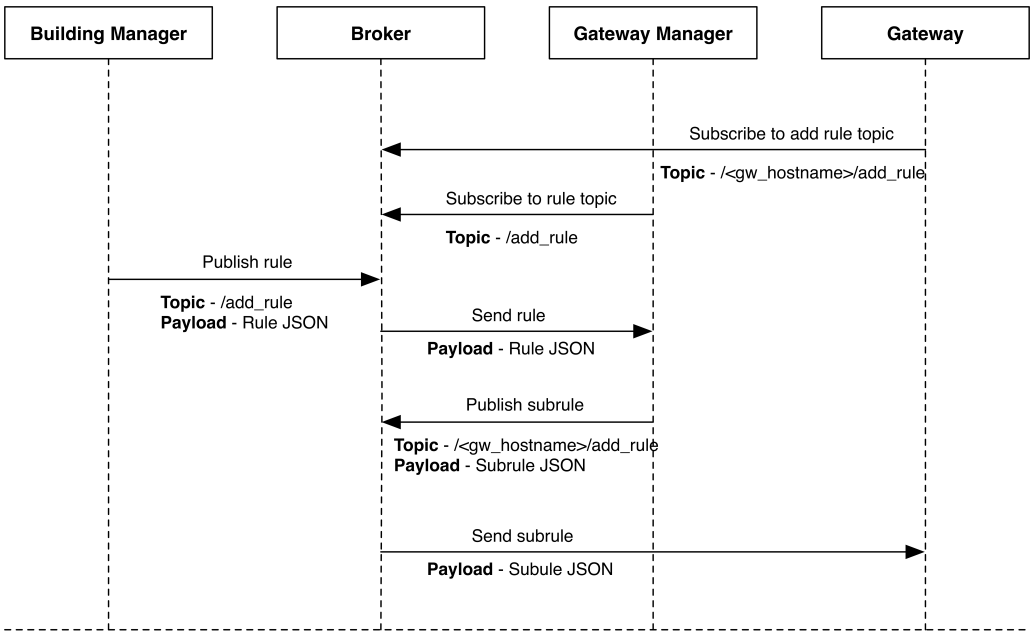

Figure 4.5: Interaction diagram of adding a new rule to the Gateway Manager and a Subrule to a Gateway.

As can be observed, the Building Manager publishes a rule and the Gateway Manager parses it in order to divide them into subrules. These subrules are numbered and then distributed to Gateways using the topic(already subscribed in the broker by Gateways) "/<gw_hostname>/add_rule".

System Awareness

The GM, being the unit that will impose the fail handling measures, needs to be aware of each component and each gateway state. In order to do that, and as addressed before, each component has to send a periodic heart beat message to the central broker, using the "/heart_beat" topic, which is subscribed by the GM. The GM then checks periodically if every component sent the heart_beat. If a component fails and it is not able to send the heart_beat, after a programmable timeout, the GM begins the implementation of failing handling measures. These features could have been implemented using Last Will feature of MQTT. Clients could configure a Last Will message, so that upon disconnecting from the broker, due to a failure, that message would be published in the broker and the other nodes would be aware that the faulty node was disconnected. This approach was not implemented so that other protocols, such as COAP(that does not support this feature) could be used, without profound changes in the application's code.

Devices Configuration

When a new gateway is connected to the system, it sends a message to the GM, informing which devices it is able to connect and then, with this information, the GM balances the available devices among the gateways. To explain further, when two gateways are able to connect to the same devices, the GM balances that devices among the two in order to maintain a equal number of devices in each one. Also, it stores for each device, the gateways that are able to connect to it. This allows that when a gateway fails, the GM has the information needed to inform a new gateway, that are able to connected to the lost devices, to take control of those devices.

Gateway Manager Dashboard

In order to visualize the state of the gateways present in the system, a dashboard was created, using a Python3 web framework, Flask[4]. In Figure 4.6 is shown the main page of this dashboard.

[4]http://flask.pocoo.org/

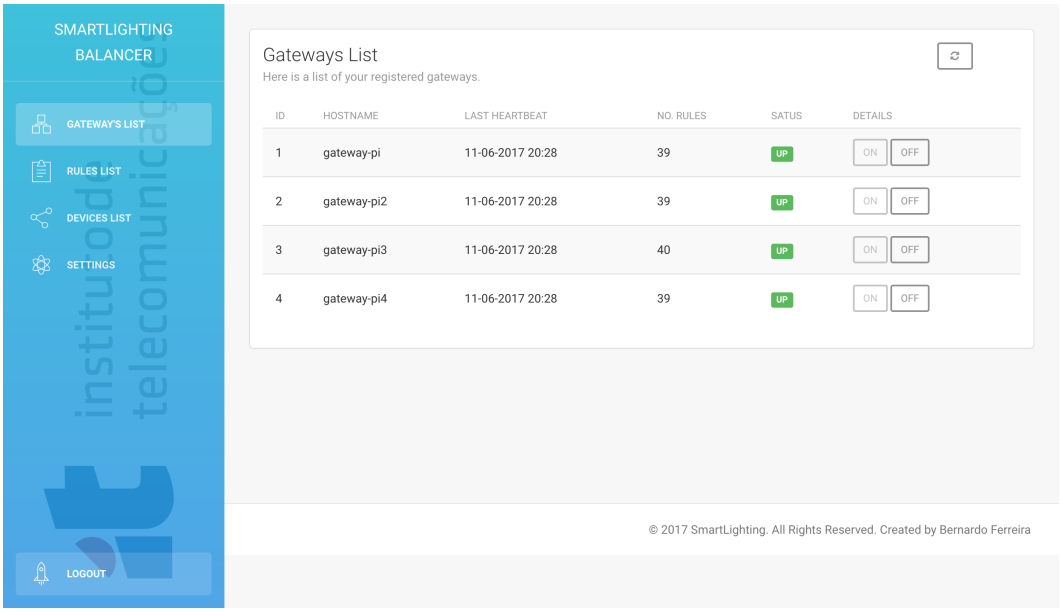

Figure 4.6: Screenshot of the Gateway Manager dashbord main page.

As can be observed, this dashboard allows to check several information about the gateways, such as, the hostnames, the time of the last heart beat received, the number of devices connected, and has also a feature to turn on/off a gateway if pleased. Also, it has not only the the list of rules and in which gateway they are being deployed, but also a list of devices and the corresponding gateway.

Gateway Failure Handling

Since Gateways are the components that communicate directly with devices, a gateway failure could have a huge impact for the building. For that reason, one of the most important features of the GM is to detect that a gateway fails and, as addressed before, inform other gateways within the range of the lost devices to assume control. Also, the rules that were present in the failing gateway, should also be sent to other gateways to be deployed. Bellow in Figure 4.7, it is represented the interaction diagram of the processes that occur when a gateway fails.

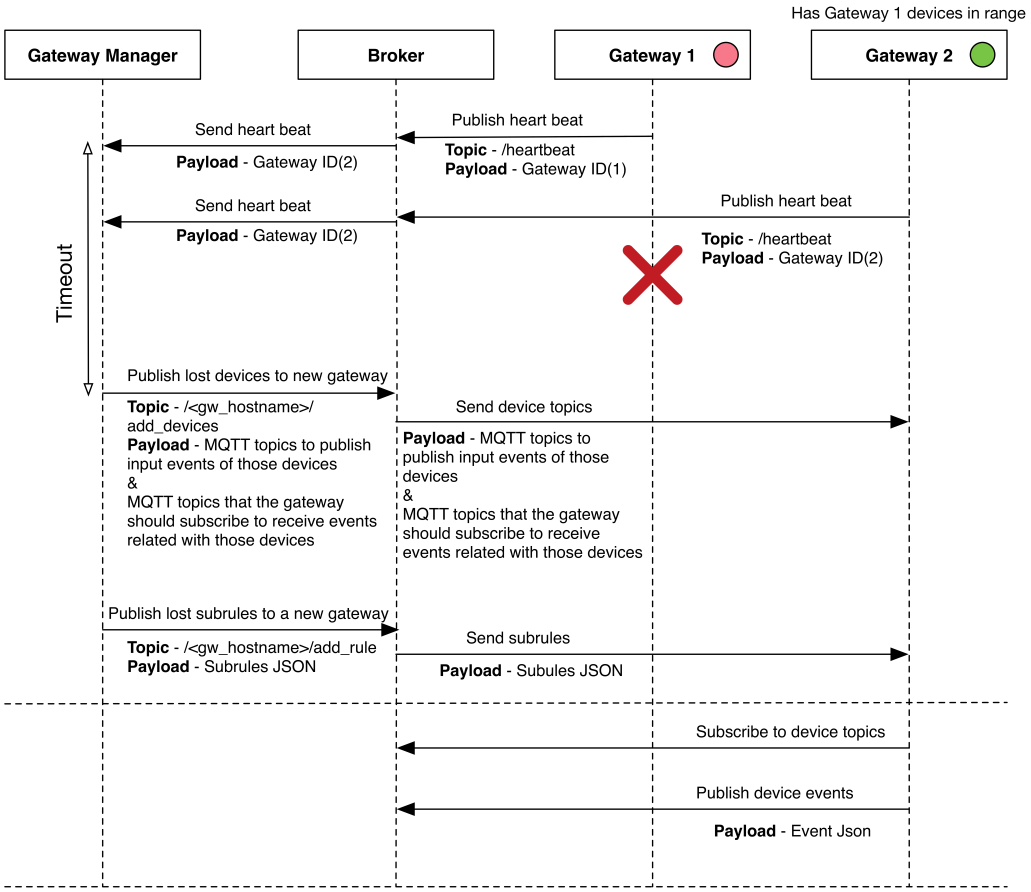

Figure 4.7: Interaction diagram of the procedures when a gateway fails.

As can be observed, since the GM has the information regarding which gateways have access to which devices, it can easily distribute the lost devices among the gateways that can communicate with them. Also, the rules that were deployed in the lost gateway are distributed through the available gateways using, again, a "round-robin" method in order to balance the number of rules in each gateway.

Gateways Synchronization

In some cases, there can be an emergency where the gateways cannot send events nor receive actions through the central MQTT Broker. In this state, Gateways must communicate directly with each others. In order to to that, not only they need to know to which gateways they should send its device events to be processed, but also need to know where they must send the processed events to. Since the Gateway Manager has the information regarding the gateways state, i.e, which rules and devices are deployed in each gateway, the GM generates, everytime there is a change in either a gateway's devices or rules, two JSON's containing this information and send them to all Gateways.

The first JSON contains a list of Listeners MQTT topics and for each one, a list of gateways that have process rules for those events. To explain further, when a device generates an event, the corresponding Gateway, uses the information in this JSON to know which gateways are deploying rules that process that event, and send it to each one of them. The second JSON contains a list of Targets and the respective gateways that have devices triggered by that output event. This way, when a gateway finishes the processing of an event, it finds in this list which gateways have devices that are triggered by that event and sends it to them.

4.5.2 GATEWAY

This section describes how the Gateway proposed features and its requirements were implemented. Since this project aims to create a smart environment in buildings, the amount of gateways needed to cover the whole network of devices may vary depending on the building size. Most solutions in the BAS systems rely on expensive gateways which can be a downside for stakeholders. The solution is to create a gateway software that is able to work in a microcontroller or in a small and cheap SoC like a Raspberry Pi[5] or a Nano Pi[6]. In an initial state, the implementation began to be done using Java, however, the program would consume too much memory, which would be a downside, taking into account the proposed requirements. With this in mind, the programming language used in the deployment of this software was MicroPython[7]. MicroPython is an open source Python programming language interpretor that runs on small embedded development boards [49] and, although being full packed with advanced features, it is compact enough run within just 256k of code space and 16k of RAM [50]. For these reasons, MicroPython was chosen to allow more flexibility when choosing gateway's hardware.

To allow the communication between gateways in a scenario where the access to the central MQTT broker might be impossible, either due to a failure in the network or in the broker itself, a Mosquitto lightweight MQTT broker was also deployed in each gateway. The use for these brokers will be explained later in section 4.6. Also, since the IP addresses of the several components, which are needed to communicate with each others, can change due to network problems, it was deployed in each node of the system an Avahi Multicast Domain Name System (mDNS) deamon[8], enabling that each component could be reached by just its hostname, which never changes for each

[5]https://www.raspberrypi.org/
[6]http://www.nanopi.org/
[7]https://micropython.org/
[8]https://www.avahi.org/

machine.

Devices Configuration and Management

When a gateway is deployed, it needs to send, to the GM a list of devices within its range and the resources that each device has. Later, the GM, responds with a list of which devices should be controlled by the gateway and in which topics should gateways publish events generated by the devices. Also, gateways receive information concerning to which topics they must subscribe in the central broker. To explain further, gateways need to know in which topic to publish events, generated by each device, in the central broker. Also, after those events being processed in the CEP Engine, gateways need to be subscribed to the output event topics that concern the devices it controls, in order to receive the automation actions. Bellow, in Figure 4.8, is represented the interaction diagram of the communications needed to configure a new device.

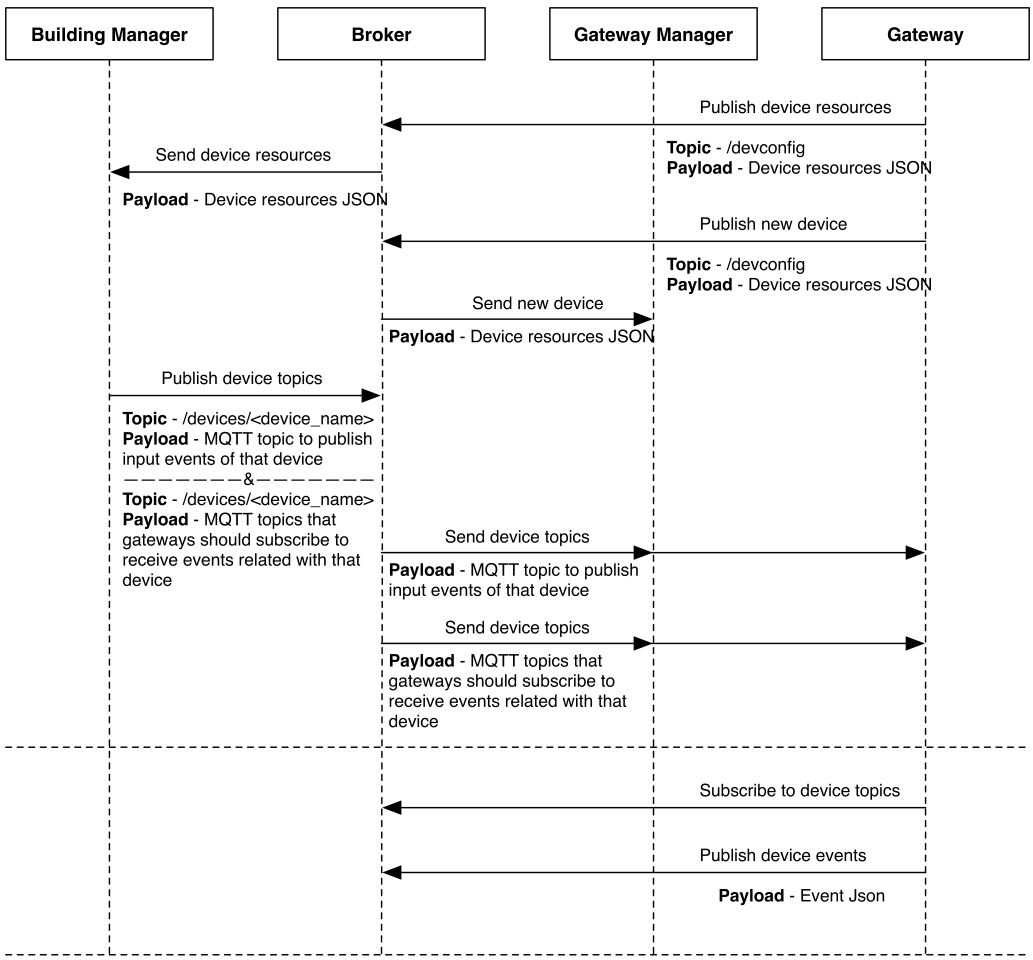

Figure 4.8: Interaction diagram of a new device configuration.

Rule Parser

The GM, as addressed before, parses the received rules and give an unique ID to each subrule. Then, when balancing the rules among the available gateways, it sends a JSON with the subrule and the given ID. When a sub rule is received in a gateway, it must be deconstructed in order for the automation engine to know which events trigger it, and what actions must be performed. In Figure 4.9 it is represented a flow diagram explaining the process of parsing a rule. As addressed before, each subrule can either have a Set Value or a Set Percent Function. In Set Value Functions, there is only one action to be parsed, however, in Set Percent Functions, there are two actions to be considered, a "Listen_value" and a "Listen_boolean" action. For each action, the parser checks the existence of the modules addressed before, i.e. Windows, Aggregators, Filters and Converters, and creates an object for each one of them, containing its informations. An Action object is then created, containing an unique ID and a reference for the modules objects. In the end, the parser saves, in a data structure, the information regarding the rule in the following way: "Rules[Listeners] = [ActionsID]", where, Rules is a list of MQTT topics, of the different listeners parsed, and each one of them has a list of ActionsIDs with the same Listener. This was implemented this way so that when an event is received, it is only needed to search for the corresponding Listener topic, in the data structure, to know which actions must be triggered.

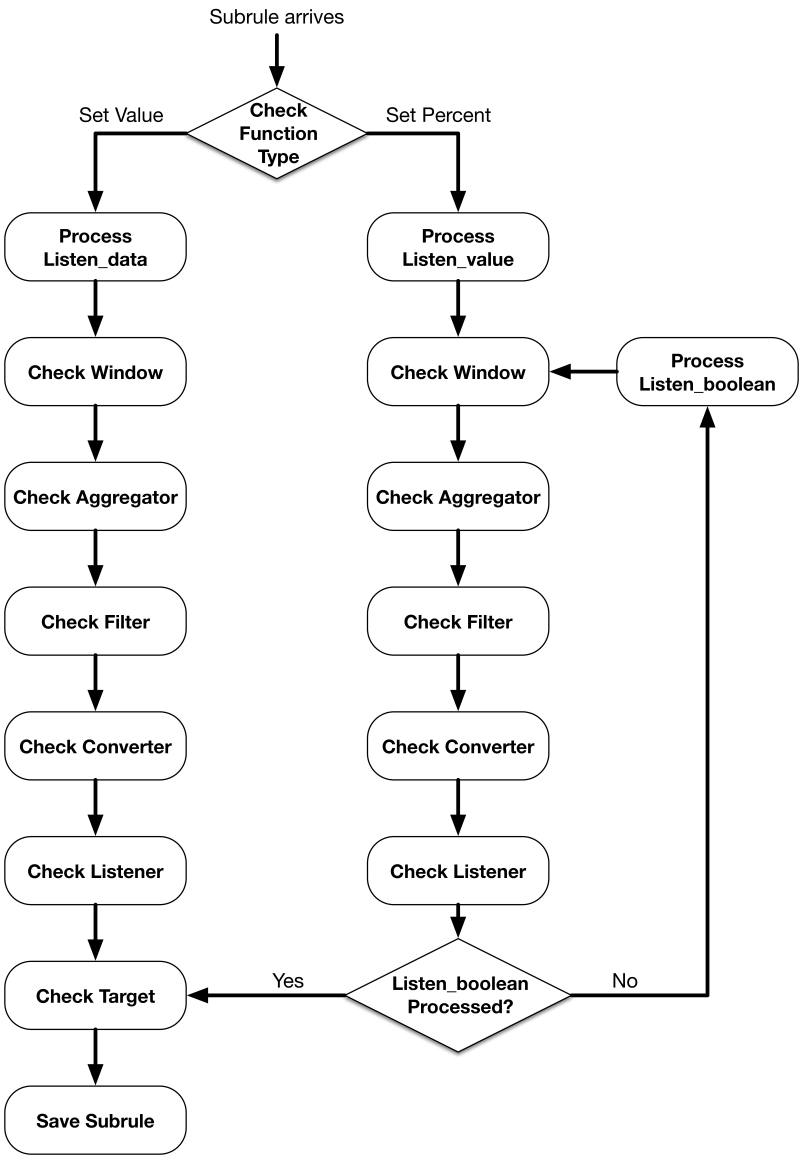

Figure 4.9: Rule parser flow diagram

Gateway Automation Engine

The Gateway Automation Engine (GAE), as stated before, will only work in emergency states, either if the central broker or the CEP Engine are down. When this happens, gateways subscribe, in the central broker, to the MQTT topics of the listeners present in the rules deployed by each one. This way, the events generated by the devices are published in the broker and then sent to gateways that have rules with those events as input.

As mentioned earlier, when an event arrives to a gateway, first it must discover the Actions that need to be triggered so that the event can be processed. The GAE was

designed to implement the modules present in Table 4.1.

Type	Module	Description
Listener	MQTT	MQTT topic to receive events
Target	MQTT	MQTT topic to send events
Function	Set Value	Set the value received from input streams to the output streams
	Set Percent	Set a percentage of a value from input streams to the output streams, based on boolean data from another input stream
Filter	Equal	Filter events with the value equal to a given value
	Not Equal	Filter events with the value different than a given value
	Greater Than	Filter events with the value greater than a given value
	Less Than	Filter events with the value less than a given value
	Greater or Equal Than	Filter events with the value greater or equal to a given value
	Less or Equal Than	Filter events with the value less or equal to a given value
Window	Time Window	Capture events in a predefined time
	Length Window	Capture a predefined number of events
Aggregator	Average	Calculate the average value of all the events in a window
	Any	Returns 1 if any of the values in the window is greater than 0
	None	Returns 1 if all the values in a window are 0
Converter	Lux To Percentage	Calculates an output percentage value to apply on a luminaire's dimming level based on a value in lux units
	Set 1	Sets the value to 1
	Set 0	Sets the value to 0

Table 4.1: Implemented modules in Gateway's automation engine.

For each action, the GAE follows certain steps, as can be visualized in Figure's 4.10 flow diagram. The first module to be applied is the Filter, if present in the action. If the value present in the event does not pass the filter verification is automatically discarded. After that, the GAE checks if the action has a window, and as consequence an aggregator, and if so, which type of window and applies it. For instance, with a Time Window with an Any Aggregator, the program must send the value 1 and, if no other event is received within the window time length, the GAE must send a value 0. If a new event is received within the time window length, the old window is discarded and begins a timer for the new one. Since micropython is a single-threaded programming language, in order to implement features like this one, it had to be used an event pool, in which, each event received is added as a new task for the event pool, and this way, to implement the time window, per example, is only needed to put the task of that event to sleep, the expected time length. Meanwhile the program can run and process other events present in the pool. The implementation of the Time Window module can is represented in Snippet 4 in Appendix B. Lastly, the Converter module, if present, processes the value based on the converting rules given and an event is sent to the central broker.

This automation engine will only be enabled if the CEP Engine or the central broker fail, as will be addressed and explained later.

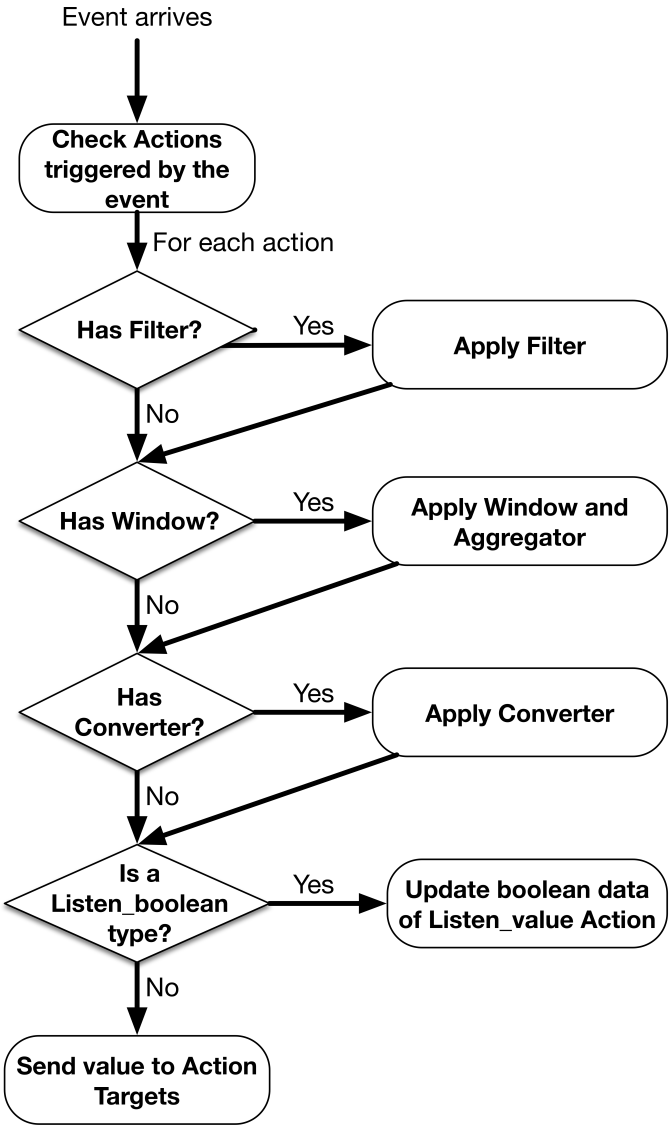

Figure 4.10: Event processing flow diagram.

4.6 OPERATIONAL SCENARIOS

As one of the most important requirements and objectives for the present dissertation project, the fail handling mechanisms are a core feature for every smart environment deployed in a building. In order to rely with assurance in a fully autonomous system, it must be adapted to answer swiftly to unexpected behaviours within the system. In the scope of this project, the most important component, to impose an autonomous system, is the CEP engine, however if that component fails, due to internal software failure or a communication failure, the integrity of the building as well as the safety of its occupants could be at risk. For this reason, as addressed before, an automation engine (GAE) was implemented so that gateways could process events in order to trigger actions.

In section 3.4, were introduced five different operational scenarios: every component working as expected, CEP Engine down, Gateway Manager down, both CEP Engine and Gateway Manager down, and lastly, the MQTT broker down. For each of these scenarios, it will be explained how the system handle each of the failures in order to maintain the minimum functionalities of the building.

Scenario 1 - All components are operational

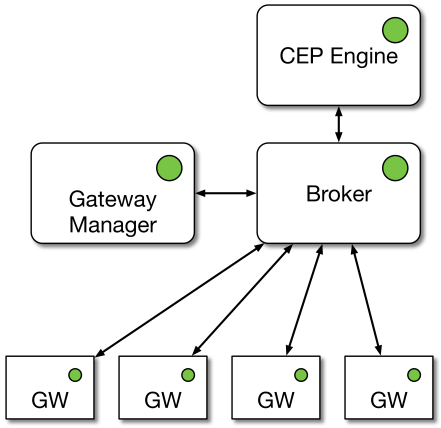

Figure 4.11: Functioning scenario 1.

In scenario 1, as can be observed in Figure 4.11, all components are working in the expected way. Devices send events to the connected gateways, which forwards them to the central MQTT Broker. Then the CEP Engine process them, based on the deployed rules and publishes the output events on the MQTT Broker. Since gateways, as seen before, subscribe to all the topics that concern their devices, they receive the actions and send them to the devices.

In this scenario, the automation engine of the gateways is disabled, since the central CEP is operational. Finally, this scenario support Gateways failure, since the Gateway Manager is also operational, as it was explained in Figure 4.7.

Scenario 2 - CEP Engine failure

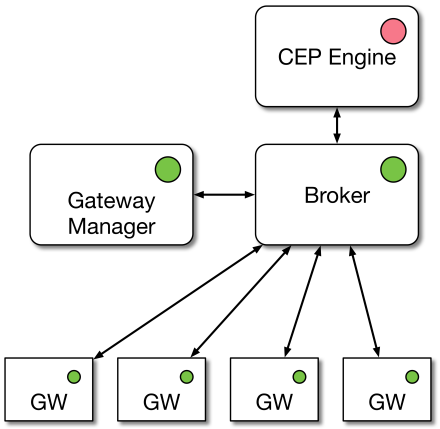

Figure 4.12: Functioning scenario 2.

In scenario 2, represented in Figure 4.12, the CEP Engine is down. As addressed earlier, when this happens, both the Gateways and the Gateway Manager stop receiving its heart beats, and after a configurable timeout (15 seconds in this implementation), gateways subscribe, in the central broker, to the MQTT topics of the listeners present in the rules deployed by each one, and start processing device events. This process is represented in the interaction diagram present in Figure 4.13

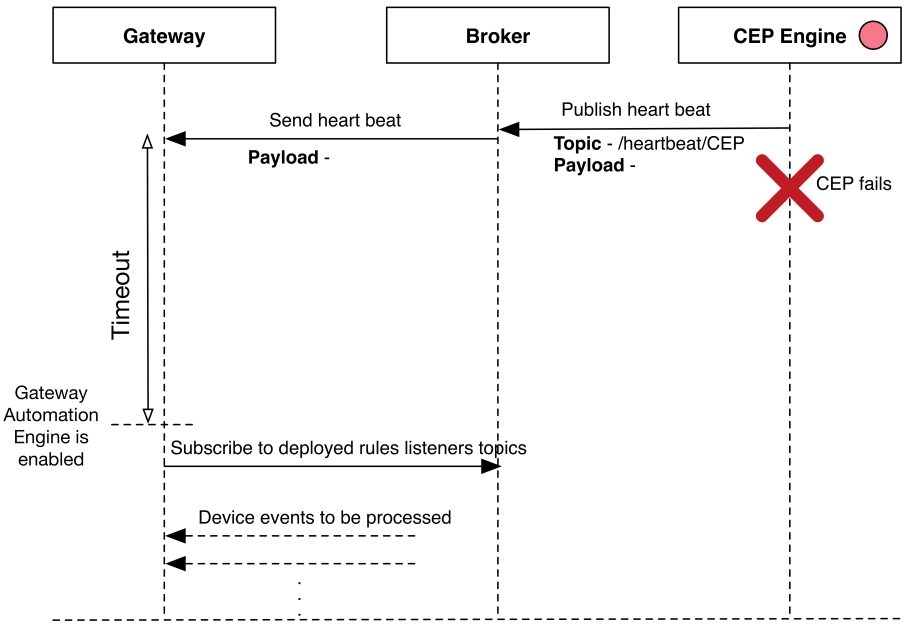

Figure 4.13: Interaction diagram of the processes that take place when CEP Engine fails.

Again, this scenario supports failures in gateways, since the Gateway Manager is operational, using the process explained earlier and represented in Figure 4.7.

Scenario 3 - Gateway Manager failure

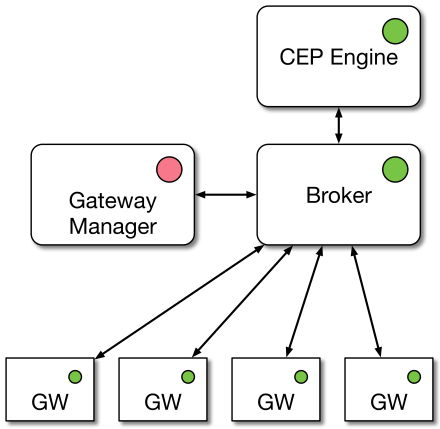

Figure 4.14: Functioning scenario 3.

Scenario 3 represents a failure in the Gateway Manager while all the other components are working correctly. When this happens, the system continues to work correctly, however if a gateway fails while in this scenario, the lost devices and rules will not redistributed to other gateways and thus, the automation of some building areas will stop working. Also, while in this state, no new rules can be added to gateways. Since none of the other components are directly dependent of the Gateway Manager, for its normal operation, its absence will not interfere with the system, and thus, the rest of the nodes can be agnostic about the Gateway Manager state.

Scenario 4 - Gateway Manager and CEP Engine failure

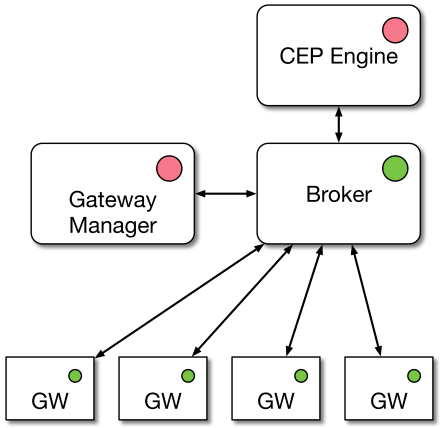

Figure 4.15: Functioning scenario 4.

In this scenario, both the Gateway Manager and the CEP Engine are down. This scenario is detected and produces the same alterations in the system as seen both in scenario 2, as the CEP Engine failure enables the processing of events in the gateways automation engine, and in scenario 3, as the Gateway Manager failure will disable gateways fail handling and rules distribution.

Scenario 5 - MQTT Broker failure

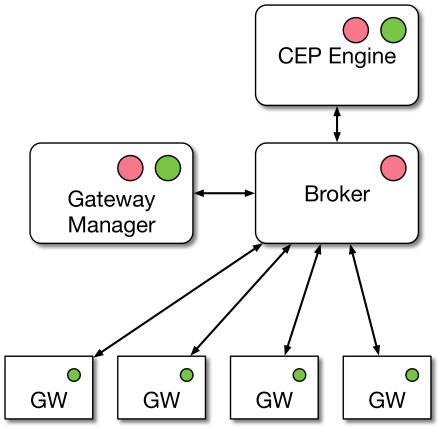

Figure 4.16: Functioning scenario 5.

This last scenario is the most severe and complex one, since if the central MQTT broker is down, the communication between gateways and the other components (GM and CEP Engine) is interrupted. When gateways are aware that the central broker is down, the automation engine is enabled, and the events generated by the devices, are sent directly to the brokers of gateways that have rules that process those events. Also, when a gateway receives an event to be processed, it sends the resulting event to gateways that have devices triggered by that event. As seen before, the Gateway Manager synchronizes, between all gateways, the information needed to implement these features: each gateway knows exactly to which gateways the events, of each device it controls, should be sent to be processed, and also, has the information of to which gateways should the processed events be sent. While in this scenario, which could be triggered by a whole building network failure, the gateways would still be able to connect through bluetooth with devices, and thus, the building's most core functionalities, such as lighting, will still be able to work.

Summary

Looking at the described scenarios, it is noticeable a degradation over the system functionalities. However, regardless the scenario, the most important automation

features, that are crucial to the system, can be safeguarded by deploying emergency rules in gateways. Even when core components of the system are down, gateways can still ensure a minimum functionality to the building by themselves. Bellow in Table 4.2, are specified the scenarios, and the component's state and distinctive features present in each one of them.

Scenarios	**Broker**	**CEP Engine**	**Gateway Manager**	***Gw Failure Handling***	***Gw Automation Engine Enabled***	***Gw-Gw Communication***
1	Up	Up	Up	Yes	No	No
2	Up	Down	Up	Yes	Yes	No
3	Up	Up	Down	No	No	No
4	Up	Down	Down	No	Yes	No
5	Down	N/A	N/A	No	Yes	Yes

Table 4.2: Scenarios summary table.

As can be observed, the gateway's Automation Engine is only enabled either on scenarios where the CEP Engine is down, or when the broker is unavailable, since it connects gateways to the CEP Engine. Also, since the Gateway Manager is the component responsible for handling gateway failures as redistribute devices and rules, when this component is down, the gateway's failure handling is not assured. Finally, in scenario 5, since the central broker is down, gateways need to communicate directly with each others in order to insure the processing of its events.

As referred before, the system should be working on scenario 1 the great majority of time, however, since all the other scenarios ensure a basic functionality to the system, a fail in a component would not impose great consequences to the system. Therefore, the implemented features addressed all the requirements and objectives for the proposed dissertation.

CHAPTER 5

Evaluation and Results

The objective for this chapter is to validate the proposed solution. Firstly, in section 5.1, a deployment scenario overview will be given, highlighting the devices and tools used for that purpose. Following, in section 5.2, the obtained results for different tests will be analysed. With the requirements, addressed in chapter 3, taken into consideration, the performance of the automation engine, implemented in gateways, will be evaluated, and also, to judge the capacity of the system to react to abnormal operational scenarios, the reaction time to this occurrences will be measured.

5.1 DEPLOYMENT SCENARIO

The final deployment scenario, used to test and validate the system and illustrated in Figure 5.2, is composed by the Building Manager platform, the WSO2 CEP Engine, the Gateway Manager and two Gateways, all communicating through the MQTT Broker.

The Building Manager is the component responsible for supporting the user-friendly platform to create rules and distribute devices through the building planer. This platform was hosted in a Linux virtual machine equipped with a single-core processor and 1 gigabyte of memory.

The WSO2 CEP engine, which is one of the core components of the whole system, as it is the primary processor of events, based on the logic specified in the rules. This component, required a more powerful machine than the Building Manager component , being hosted in a Linux virtual machine with a quad-core processor and 4 gigabytes of memory.

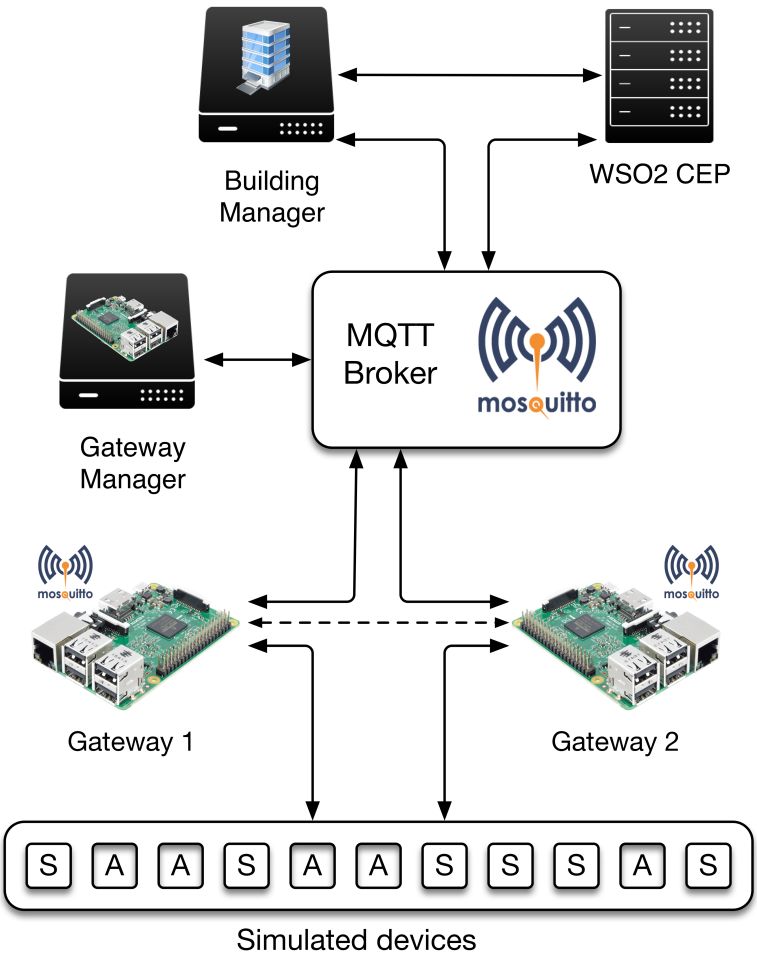

Figure 5.1: Deployment Scenario.

Regarding the Gateway Manager, it contains both a software for gateway's management and a web-server to offer a platform to check the state of each gateway and the general state of the system. This component was also deployed in a Linux virtual machine with a dual-core processor and 2 gigabytes of memory.

In order to assure the communication between the different components, the Mosquitto MQTT Broker was deployed in a separate virtual machine with a dual-core processor and 2 gigabytes of memory.

Finally, concerning the gateways, which are responsible for communicate with devices and, in some scenarios, process events using its Automation Engine, it was used a Raspberry Pi 3 Model B, which has a quad-core processor with 1 gigabyte of memory [1].

Regarding the deployment scenario, a equal set of simulated devices were configured in both gateways, simulating a scene where the devices are in range of two different gateways. Each gateway was equipped with code to generate events of the devices assigned to them, by the Gateway Manager. If a gateways had control over a sensor, it should send events with a fixed programmable frequency. This was used for the testing scenario addressed in section 5.2.2.

All the tests were done using the internal network of IT Building.

5.2 PERFORMANCE RESULTS

In this section will be presented the results and discussion of the tests preformed. Firstly, in section 5.2.1 the latency of the gateway's Automation Engine will be mesured and then, in section 5.2.2, the reaction time to system failures will be evaluated.

5.2.1 GATEWAY'S AUTOMATION ENGINE PERFORMANCE

In order to evaluate the gateway's Automation Engine performance, a stress test was preformed. The objective was to measure the processing time of events, in five different stress situations: first, sending a single event and measuring the time taken by the engine to produce an output response, and then increasing the number of consecutive events, 10, 100, 1000 and 10000, to not only measure the time taken to process all those events, but also to check if all events were processed.

For this test, two python scripts were made: one to send a programmable number

[1] https://www.raspberrypi.org/products/raspberry-pi-3-model-b/

of events consecutively, and other to measure the time since the first event was sent, and the last response action was received. The deployed rule consisted in inputting a motion event and respond with an event to turn on a light as result. The results of this test are expressed in Figure 5.2.

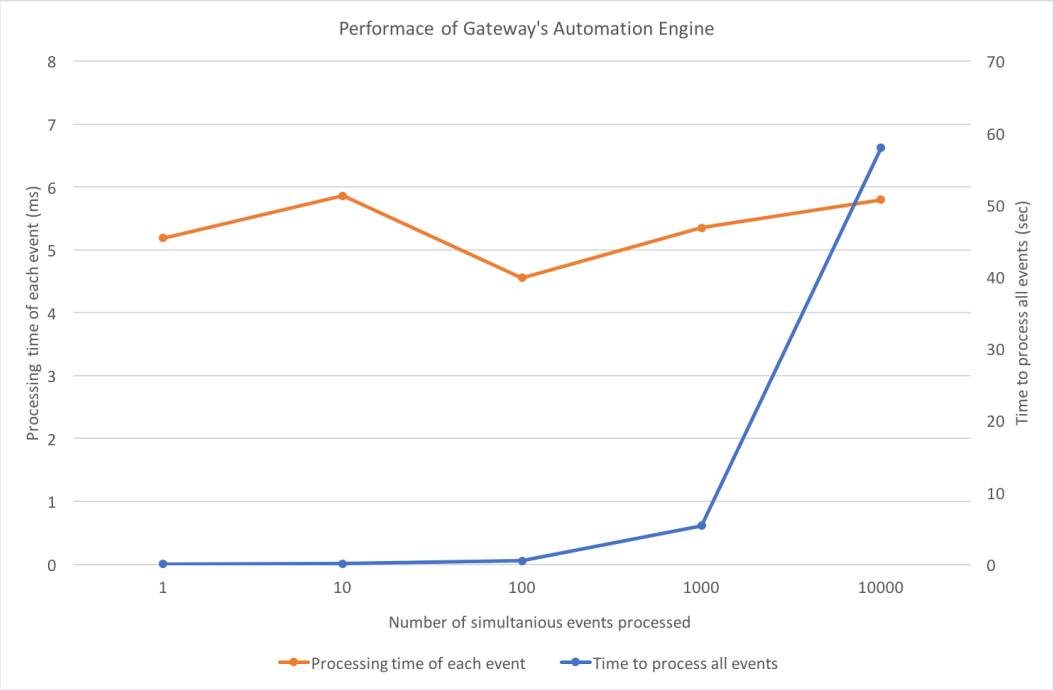

Figure 5.2: Gateway's Automation Engine Performance.

As can be observed, the time taken to process each event in the different test scenarios was, in average, between 4 to 6 milliseconds, which is about the same time taken by the WSO2 CEP Engine, tested in a previous dissertation [2], when processing simple rules like the one used for this test. Bellow in table 5.1 are shown more detailed insights about this test.

	1	**10**	**100**	**1000**	**10000**
Average Time	5,2 ms	5,9 ms	4,6 ms	5,3 ms	5,8 ms
Minimum Time	4,4 ms	5,6 ms	4,5 ms	4,5 ms	5,5 ms
Maximum Time	6,0 ms	6,3 ms	4,6 ms	6,7 ms	6,1 ms
Standard Deviation	0,7 ms	0,3 ms	0,1 ms	0,9 ms	0,3 ms
Total Time	0,005 sec	0,059 sec	0,455 sec	5,350 sec	57,953 sec

Table 5.1: Time taken to process each event.

This test was performed 10 times for each number of simultaneous events. Regarding the time to process each event, it was expected that when increasing the number of simultaneous events, the time needed to process each one of them would increment,

since it would increase the processor load, however, as can be observed in Figure 5.2, there is a slight decrease for 100 simultaneous events. This can be explained by the internal architecture of Raspberry Pi's processor and cache.

Raspberry 3 Performance

One of the objectives for this dissertation was to built a Gateway software able to operate in constrained devices, such as a Raspberry Pi. While doing the stress test above, it was measured the amount of CPU and memory used by the implemented software. In Figures 5.3 and 5.4, is represented the output of "htop"[2] command, which is an interactive process viewer for Unix, before and while the processing of the 10000 events, respectively.

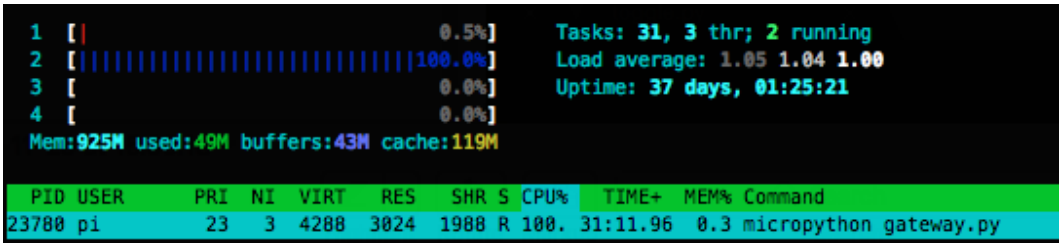

Figure 5.3: Output of "htop" command with no events being processed.

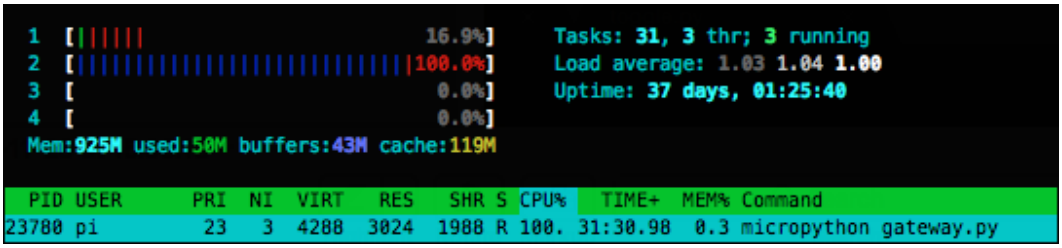

Figure 5.4: Output of "htop" command while processing 10000 events.

As can be observed, not only the gateway software only uses a small amount of memory, but also, the CPU usage while processing the 10000 events versus while in idle, only increases the usage of the second core by 16%. Concluding, the use of Micropython as the chosen language for this software, paid off by presenting great results in terms of memory and CPU usage. This means that other SoC's, with lower specifications and even lower power consumptions, could be used in this project.

[2]https://hisham.hm/htop/

5.2.2 SYSTEM FAILURE'S REACTION PERFORMANCE

In order to test the system reaction to fail scenarios, two tests were done. The first one was to measure the time needed, after a failure in the CEP engine or in the MQTT Broker, for gateways to enable its automation engine and start processing events. This test addresses the reaction time to scenarios 2,4 and 5, addressed in Chapter 4.6. In these three scenarios, the heartbeat message from the CEP Engine (which that states that this component is running, and thus, there is no need for the gateway's automation engine to be running) stop reaching gateways, either because the CEP Engine actually failed (scenarios 2 and 4), or because the MQTT Broker failed (scenario 5). The scenarios are different, however, the reaction to them is the same for gateways: the gateway's automation engine is enabled and starts processing events.

In order to obtain reaction time of gateways to these scenarios, a Python script was made to measure the time since the central CEP Engine went down (purposely) and the receiving of the first gateway processed result event. Also, input events were being sent to the broker, every 100 milliseconds, this way, after the gateway automation engine was enabled it would receive an input event to process within 100 ms. The obtained results can be consulted bellow, in table 5.2.

Average Time	7,261 sec
Minimum Time	5,084 sec
Maximum Time	8,883 sec
Standard Deviation	1,185 sec

Table 5.2: Reaction time to a CEP Engine failure

As can be observer, in average, a gateway needed 5 to 9 seconds to react to the CEP engine failure. Note that gateways wait for five seconds (configurable timeout) before conclude that the CEP Engine is down and thus, enable its own automation engine to process events.

Regarding scenario 3, in which there is a failure in the Gateway Manager, no reaction times can be measured, since the failure of this component does not produce any alteration to the normal operation of the system, i.e. the processing of sensor's generated events.

The second test objective was to measure the time needed to recover from a gateway failure. When a gateway fails, the devices that are in its control become unreachable and thus, the events generated are not published in the broker. The gateway manager, then, needs to check which other gateways have access to those devices and assign them to a new gateway. Also the rules that were deployed in the failing gateway need to be

sent to other working gateways. For this test in specific, the same device was configured in two different gateways, and, a event simulator was also implemented in each one of them in order to gateways publish simulated events of the devices they control. To explain further, two gateways were connected with access to the same device, and the Gateway Manager chose one of them to control that device. Then, using a Python script, it was measured the time since the gateway, that was in control of the device, was disconnected, until the other gateway, took over the device and started publishing events. The results of this test were as shown in table 5.3

Average Time	13,194 sec
Minimum Time	7,492 sec
Maximum Time	18,877 sec
Standard Deviation	3,839 sec

Table 5.3: Reaction time to a Gateway failure.

As can be concluded, by observing the results, the reaction time of a gateway failure is in average 7 to 19 seconds. Note that in this test, gateways were sending heart beats in 5 second intervals, and the the GM was checking every 10 seconds whether a gateway was disconnected for more than 10 seconds. That is why there is such discrepancy between the maximum and minimum values obtained. It depends on when the last heart beat of the failing gateway was sent.

Then, using the same scenario, the disconnected gateway was put back up, and since it was a known gateway for the Gateway Manager, immediately after the first heart beat was received, it attributed the control of the same device back to that gateway. The results, as can be seen in table 5.4, were from 129 to 199 milliseconds.

Average Time	0,164 sec
Minimum Time	0,129 sec
Maximum Time	0,199 sec
Standard Deviation	0,024 sec

Table 5.4: Reaction time to a Gateway back up.

Summary

To summarize, given these results, the Gateway's Automation Engine is proven to be adequate to be implemented as a back up plan for emergency states where the CEP Engine might not be reachable. Also since the system is able to react to component's failures within a reasonable time, the requirements and objectives proposed were successfully implemented.

CHAPTER 6

Conclusions and Future Work

This final chapter aims to provide an overview of the developed solution. As it was referred in chapter 4 and later validated in chapter 5, the proposed solution fulfilled all the projected requirements and objectives.

Nevertheless, during the development of this solution some issues were found. First, the looking at the scenarios presented in chapter 3, the idea was to, regardless the state of the system, Gateways being able to recover from failures, and for that reason, the Gateway Manager was developed. However, since this component is dependent on the MQTT Broker, and it can fail itself, there are some specific cases where the handling of a gateway failure is not assured. The solution for this issue would have been possible if gateways could identify that other gateway had failed and, by communication with each other, elect a new gateway to take over the control for the lost devices and rules. Also, the Gateway Manager did not need to synchronize the information, between gateways, concerning to where each gateway must send its device events in case of a failure in the MQTT Broker. A service discover application like Avahi, could be used for each gateway broadcast the devices and rules they control, and this information could be accessible by any other gateway. An approach like this one could resolve the Gateway Manager dependence, being this component used only to distribute devices, rules and provide a platform to control gateways, however, this solution would introduce more load in the network. It would be interesting for the project, to test this approach in a future work. Also, persistence measures should be implemented in both gateways and in Gateway Manager component, so that after a failure, the process of recover from it could be faster.

Another feature left for future work was the User Management platform that, although it was not in the scope for this dissertation, would be interesting for Smart-Lighting project's purpose.

Regarding security aspects, although its undeniable importance, were not considered in the implementation. This should also be addressed in a future work, by, for instance, encrypt the payload of MQTT messages and user authentication to the Building Management platform.

References

[1] I. Nolte and D. Strong, *Europe's Buildings Under the Microscope*. 2011.

[2] H. Moreira, "Sensor Data Integration and Management of Smart Environments", 2016.

[3] M. Brambley, P. Haves, P. Torcellini, and D. Hansen, "Advanced Sensors and Controls for Building Applications : Market Assessment and Potential R & D Pathways", *Pacific Northwest National Laboratory*, no. April, p. 162, 2005.

[4] *Johnson Controls Tyco merger drives smart building security :: News - American Dynamics | Security companies - SourceSecurity.com*. [Online]. Available: `https://www.sourcesecurity.com/companies/micro-site/american-dynamics/news/tyco-johnson-controls-merger-smart-building-security-co-3094-ga-co-10023-ga-co-2900-ga-co-3481-ga-co-2554-ga-co-10181-ga-co-8421-ga-sb.19479.html` (visited on 06/22/2017).

[5] *BACnet Website*. [Online]. Available: `http://www.bacnet.org/` (visited on 06/26/2017).

[6] *KNX Association - KNX Association [Official website]*. [Online]. Available: `https://www.knx.org/pt/` (visited on 06/26/2017).

[7] Echelon Corporation, "Introduction to the LonWorks® Platform", pp. 1–98, 2009.

[8] *DALI - Digital Addressable Lighting Interface*. [Online]. Available: `http://www.dali-ag.org/` (visited on 06/26/2017).

[9] A. Iwayemi, W. Wan, and C. Zhou, "Energy Management for Intelligent Buildings", in *Energy Management Systems*, P. G. Kini, Ed., Rijeka: InTech, 2011, ch. 06.

[10] Siemens, "Communication in building automation", p. 14, 2013.

[11] C. Reinisch, W. Kastner, G. Neugschwandtner, and W. Granzer, "Wireless technologies in home and building automation", *IEEE International Conference on Industrial Informatics (INDIN)*, vol. 1, pp. 93–98, 2007.

[12] M. Weiser, "The computer for the 21st century", *Scientific American (International Edition)*, vol. 265, no. 3, pp. 66–75, 1991.

[13] K. Ashton, *That 'Internet of Things' thing*, 2009.

[14] D. Evans, "The Internet of Things - How the Next Evolution of the Internet is Changing Everything", *CISCO white paper*, no. April, pp. 1–11, 2011.

[15] A. Al-fuqaha, S. Member, M. Guizani, M. Mohammadi, and S. Member, "Internet of Things: A Survey on Enabling Technologies, Protocols, and Applications", vol. 17, no. 4, pp. 2347–2376, 2015.

[16] R. Tatum, *Where Does the Building Automation System Fit In An Internet of Things World?* [Online]. Available: http://www.facilitiesnet.com/buildingautomation/article/Where-Does-the-Building-Automation-System-Fit-In-An-Internet-of-Things-World-Facilities-Management-Building-Automation-Feature--16651 (visited on 10/29/2017).

[17] IEC, "Internet of Things : Wireless Sensor Networks Executive summary", p. 78, 2014.

[18] M. Bellis, *The History of WiFi (Who Invented It?)*, 2017. [Online]. Available: https://www.thoughtco.com/who-invented-wifi-1992663 (visited on 09/21/2017).

[19] Bluetooth SIG, *Our History | Bluetooth Technology Website*, 2016. [Online]. Available: https://www.bluetooth.com/about-us/our-history (visited on 09/21/2017).

[20] N. S. Kemp, I. Rasool, and A. H., "Overview of the IEEE 802.15.4 standards family for Low Rate Wireless Personal Area Networks", *2010 7th International Symposium on Wireless Communication Systems*, pp. 701–705, 2010.

[21] K. Devadiga, "IEEE 802.15.4 and the Internet of things", pp. 4–7, 2003.

[22] Zikrillah, *OSI Model*. [Online]. Available: https://github.com/zikrillah/F5-Blueprint-Material/wiki/OSI-Model (visited on 09/22/2017).

[23] A. Chalappuram, P. Sreesh, and J. M. George, "Development of 6LoWPAN in Embedded Wireless System", *Procedia Technology*, vol. 25, no. Raerest, pp. 513–519, 2016.

[24] *6LoWPAN Wireless Connectivity Overview*. [Online]. Available: http://www.ti.com/lsds/ti/wireless-connectivity/6lowpan/overview.page (visited on 09/23/2017).

[25] U. Mehboob, Q. Zaib, and C. Usama, "Survey of IoT Communication Protocols Techniques, Applications, and Issues", 2016.

[26] INSTEON, "Insteon Whitepaper: Compared", pp. 0–45, 2013.

[27] A. Koubaa, M. Alves, and E. Tovar, "IEEE 802.15.4: a Federating Communication Protocol for Time-Sensitive Wireless Sensor Networks", 2017.

[28] M. Andersson, "Use case possibilities with Bluetooth low energy in IoT applications Use case possibilities with Bluetooth low energy in IoT applications -White paper", p. 16, 2014.

[29] Wi-Fi Alliance, *Wi-Fi HaLow*, 2017. [Online]. Available: https://www.wi-fi.org/discover-wi-fi/wi-fi-halow (visited on 11/20/2017).

[30] E. B. Turan, *MQTT Nedir? IoT ile Bağlantısı Nedir? - Eczacıbaşı Bilişim*. [Online]. Available: https://www.ebi.com.tr/blog/mqtt-nedir-iot-ile-baglantisi-nedir/ (visited on 09/25/2017).

[31] Y. Chen and T. Kunz, "Performance evaluation of IoT protocols under a constrained wireless access network", *2016 International Conference on Selected Topics in Mobile and Wireless Networking, MoWNeT 2016*, 2016.

[32] *MQTT Essentials Part 6: Quality of Service Levels*. [Online]. Available: http://www.hivemq.com/blog/mqtt-essentials-part-6-mqtt-quality-of-service-levels (visited on 09/25/2017).

[33] T. Salman and R. Jain, "Networking Protocols for Internet of Things", pp. 1–28, 2013.

[34] C. Lerche, K. Hartke, and M. Kovatsch, "Industry adoption of the Internet of Things: A constrained application protocol survey", *IEEE International Conference on Emerging Technologies and Factory Automation, ETFA*, 2012.

[35] N. Badugu, *Internet of Things (IoT) Application Protocols*. [Online]. Available: https://www.linkedin.com/pulse/internet-things-iot-application-protocols-narsimhmaswamy-badugu (visited on 09/26/2017).

[36] Transformative Wave, "How the IoT is Reshaping Building Automation", pp. 1–14,

[37] *Intel® Internet of Things Solutions Alliance: Overview*. [Online]. Available: https://www.intel.com/content/www/us/en/partner/solutions-alliance/program-overview.html (visited on 09/27/2017).

[38] Intel, "Affordable Building Automation Systems Enabled by the Internet of Things (IoT)", 2017.

[39] Tutorials Point, *Component-Based Architecture*. [Online]. Available: https://www.tutorialspoint.com/software%7B%5C_%7Darchitecture%7B%5C_%7Ddesign/component%7B%5C_%7Dbased%7B%5C_%7Darchitecture.htm (visited on 10/25/2017).

[40] J. Sun, X. Dong, X. Zhang, W. Gong, and Y. Wang, "High availability analysis and evaluation of heterogeneous dual computer fault-tolerant system", *Proceedings of the IEEE International Conference on Software Engineering and Service Sciences, ICSESS*, pp. 460–464, 2014.

[41] M. Zaiter, S. Hacini, and Z. Boufaida, "Agents for Fault Detection and Fault Tolerance in Component-based Systems", no. c, pp. 42–47, 2013.

[42] R. V. Kshirsagar and A. B. Jirapure, "A Survey on Fault Detection and Fault Tolerance in Wireless Sensor Networks", *International Journal of Computer Applications*, pp. 6–9, 2012.

[43] M. Oriol, M. Wahler, E. Ferranti, T. Gamer, and T. de Gooijer, "Fault-tolerant Fault Tolerance for Component-based Automation Systems Categories and Subject Descriptors", pp. 49–58,

[44] E. Redwine and J. L. Holliday, *Reliability of Distributed Systems*. [Online]. Available: http://www.cse.scu.edu/%7B~%7Djholliday/REL-EAR.htm (visited on 10/28/2017).

[45] T. N. Gia, A. M. Rahmani, T. Westerlund, P. Liljeberg, and H. Tenhunen, "Fault tolerant and scalable IoT-based architecture for health monitoring", *SAS 2015 - 2015 IEEE Sensors Applications Symposium, Proceedings*, 2015.

[46] H. Moreira, G. Correia, M. Silva, A. Marques, J. Barraca, L. Alves, P. Fonseca, and N. Lourenço, "SmartLighting platform for intelligent building management", pp. 227–238, 2016.

[47] WSO2, *WSO2 Complex Event Processor*, 2017. [Online]. Available: https://wso2.com/products/complex-event-processor/ (visited on 10/29/2017).

[48] IPSO Alliance, "IPSO SmartObject Guideline - Smart Objects Starter Pack1.0", pp. 1–39, 2014.

[49] *MicroPython Basics: What is MicroPython?* [Online]. Available: https://learn.adafruit.com/micropython-basics-what-is-micropython/overview (visited on 10/03/2017).

[50] *MicroPython - Python for microcontrollers*. [Online]. Available: https://micropython.org/ (visited on 10/03/2017).

Appendix A: Rule example

```
{
    "name": "Labs Lights",
    "subrules": [{
        "actions": [{
            "target": {
                "type": "mqtt",
                "topic": "/SM/out_events/IT2/Floor_1/Lab/1.10/1/*/3302/*/5851/*",
                "value_type": "int"
            },
            "function": {
                "name": "setif_value_percent",
                "listen_boolean": {
                    "type": "single",
                    "listeners": [{
                        "type": "mqtt",
                        "topic": "/SM/IT2/Floor_1/Lab/1.10/1/+/3302/+/5500/+",
                        "value_type": "int"
                    }],
                    "window": {
                        "type": "time",
                        "units": "seconds",
                        "value": 6
                    },
                    "aggregator": {
                        "type": "any"
                    }
                },
                "listen_value": {
                    "type": "single",
                    "listeners": [{
                        "type": "mqtt",
                        "topic": "/SM/IT2/Floor_1/Lab/1.10/1/+/3301/+/5700/+",
                        "value_type": "float"
                    }],
                    "window": {
                        "type": "length",
                        "value": 5
                    },
                    "aggregator": {
                        "type": "avg"
                    },
                    "converter": {
                        "type": "lux_to_percentage",
                        "value": 50
                    }
                },
                "percent_if_true": 100,
                "percent_if_false": 50
            }
        }]
    }]
}
```


Snippet 3: Example of a JSON rule outputted be the Building Manager platform.

Appendix B: Gateway Automation Engine - time window module code

```
if self.window != None and self.aggregator != None:

    if self.window._type == 'time':
        if self.aggregator._type == 'any':
            if value is 0:
                return

            event_id = Action.new_event(self)

            Action.apply_converter(self, value, client,r_id, enable)

            await asyncio.sleep(self.window.value)

            if Action.events[self] == event_id:
                Action.apply_converter(self, 0, client, r_id, enable)
```

Snippet 4: Implementation of the Window Module